6-17-2005

Managing Access Control in Virtual Private Networks

Twum Djin
Dartmouth College

Managing Access Control in Virtual Private Networks

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Bachelor of Arts

in

Computer Science

by

Twum Djin

Dartmouth College

Technical Report TR2005-544
Department of Computer Science,
Dartmouth College
June 17, 2005

Abstract

Virtual Private Network technology allows remote network users to benefit from resources on a private network as if their host machines actually resided on the network. However, each resource on a network may also have its own access control policies, which may be completely unrelated to network access. Thus users' access to a network (even by VPN technology) does not guarantee their access to the sought resources. With the introduction of more complicated access privileges, such as delegated access, it is conceivable for a scenario to arise where a user can access a network remotely (because of direct permissions from the network administrator or by delegated permission) but cannot access any resources on the network. There is, therefore, a need for a network access control mechanism that understands the privileges of each remote network user on one hand, and the access control policies of various network resources on the other hand, and so can aid a remote user in accessing these resources based on the user's privileges.

This research presents a software solution in the form of a centralized access control framework called an Access Control Service (ACS), that can grant remote users network presence and simultaneously aid them in accessing various network resources with varying access control policies. At the same time, the ACS provides a centralized framework for administrators to manage access to their resources. The ACS achieves these objectives using VPN technology, network address translation and by proxying various authentication protocols on behalf of remote users.

Keywords: Virtual Private Network, Access Control, Network Address Translation, and Decentralized Delegation

Acknowledgements

First, I would like to acknowledge Professor Sean Smith, and all his support throughout this research. Professor Smith was instrumental in helping me identify a topic and develop the ideas that have lead to this work. His optimism has been inspirational, especially when I was not sure how to progress with my research.

I owe much gratitude to the other members of my thesis committee as well: Doug McIlroy and Tristan Henderson. Both were patient enough to listen to my proposals and their suggestions were most helpful in refining this research. I appreciate the time they took in reading over each revision of this document, as well as their comments about both the writing and content.

I must acknowledge the support of members of the Dartmouth Greenpass Project team: Punch Taylor, Bob Brentrup, Christopher Masone, Kwang-Hyun Baek, and Meiyuan Zhao, as well as Professor Smith's students and staff in the Dartmouth PKI Lab. These people have been very accommodating, and provided me with plenty of advice concerning my research. I thank John Marchesini, another of Professor Smith's students, who dedicated a good deal of his time to helping me configure and test installations that I needed for the project.

I must also thank Nicholas Goffee; whose work served as the inspiration for this research. His thesis document on the Greenpass Client Tools served as a guideline for this document, and I owe much of the structure and organization to the good work he did on his thesis.

I am also grateful to Jacco De Leeuw and members of the Xelerance team for being responsive to my inquiries into issues related to VPN services and setting up my own VPN server.

Much gratitude is owed to all my friends here at Dartmouth and in other colleges across the country. My friends have been most supportive of me during this research. I owe so much to the love and support of Denise Twum and Kwasi Ohene-Adu here in States, and the rest of my family back home.

My greatest gratitude goes to God who has made this work possible from start to finish; He has been my strength and wisdom throughout my life and especially in my career at Dartmouth. None of this work would be possible without Him.

Contents

1. Introduction **1**

1.1 The Big Picture…….. 1

1.2 Motivation……. 13

1.3 Goals of the ACS project...…… 17

1.4 Organization, Style and Formatting of this Document..……. 18

1.5 Related Documents…… 20

2. Background **21**

2.1 Virtual Private Networks……... 21

2.1.1 Definitions……... 21

2.1.2 IPSec…….. 24

2.1.3 PPTP…… 26

2.1.4 L2TP…… 27

2.1.5 Comparisons…… 28

2.1.6 SSL…….. 29

2.1.7 SSL VPNs…….. 29

2.2 Network Address Translation…… 33

2.2.1 Problems associated with NAT…….. 33

2.2.2 Practical applications of NAT……. 34

2.3 Linux Packet Routing…… 36

2.3.1 Definitions…….. 36

2.3.2 iptables…… 37

2.4 The Greenpass Project……... 40

2.4.1 Greenpass RADUIS Server……. 43

2.4.2 Greenpass Client Tools…… 43

2.4.3 Greenpass guest access procedure……... 46

3. The Access Control Service **47**

3.1 Objectives..…….. 47

3.2 The Model…… 48

3.3 Implementation…….. 50

3.3.1 VPN Service……. 52

3.3.2 Accounts Manager……... 55

3.3.3 The Mediator……... 57

3.3.4 Specifying Policies…….. 61

3.3.4.1 The ACS Grammar..…… 63

3.3.4.2 The Web Service..…… 67

3.4 Putting it together…….. 69

3.5 Summary…… 71

3.6 Concluding Remarks.……. 72

4. Discussions **73**

4.1 Delegation…….. 73

4.1.1 Adding RADIUS-Server support..…..……. 74

4.1.2 Adding delegated-access support..…….. 75

4.2 Usability and Efficiency…… 77

4.2.1 Handling larger client load……. 77

4.2.2 Increasing the protocol working set…… 78

4.3 Security Considerations for the ACS Framework……. 78

4.3.1 Securing access to the ACS…… 79

4.3.2 Auditing networking activity…….. 81

4.3.3 Concerns about binding IP addresses to privileges……. 81

4.4 Summary…… 84

5. Related Works **85**

5.1 Napoleon Tools…….. 85

5.2 Integrated Secure Communications System (ISCS) Project…….. 85

5.3 Globus Project…… 87

5.4 Now User Filtering Works (NuFW) Project…….. 88

6. Conclusion **90**

Appendix A

The Complete ACS Grammar……. **91**

Glossary……..... **94**

Bibliography…… **103**

Chapter 1

Introduction

1.1 The Big Picture

Today, the world is considered a global village because of the extensive access the Internet provides. Without the Internet, computer networks would be isolated clusters of interconnected machines. The Internet provides communication between these clusters, or Local Area Networks (LANs). However, connectivity comes at a price; without secure measures in place to check access to networks, anybody and everybody would have access, including malicious network users looking to corrupt data, try out a new virus or worm, or simply steal private information. Access control can be defined as all those security measures that dictate which users can or cannot gain access to a network, a device on the network, or an application (a program) on a device on the network. A network is said to be *private* when access control restricts which network users can or cannot access it. Typically, for a network to remain private the devices on the network must be accessible only via a few administrative machines generally called

gateways[1]; devices on a private network are connected to the rest of the Internet only through these gateways. Similarly devices on the Internet (outside the network) can only access network resources through these gateways.

Broadly speaking, a *network resource* is anything on a network that a user may want to gain access to - a web server, a printer, a file system, an application, etc. A network user who wishes to access resources on a private network must satisfy one or more criteria before the gateway would grant her access. The set of criteria that the user must satisfy is specified by the *access control policy* of the private network. The policy might state that the user's machine (called a *host*) should have a specific IP address (or one of a range of IP addresses), or that the user should possess some credential (such as a *certificate*) or be able to prove knowledge of some secret (a password or private key). The process by which the gateway uses its access control policy to determine who a network user is, in order to determine if the user should be allowed to access the private network, is called *authentication*. Unfortunately, there is no uniform way by which a user must authenticate with a gateway. While there are standards for authenticating called *protocols*, there are a myriad of these. Furthermore, most of those protocols require software support on the network user's machine.

[1] A gateway is not restricted to secure or private networks; in most LAN setups, not all devices on the network have direct access to the Internet. Instead, there are a few devices that do (gateways) have direct access to the Internet, and all other devices that do not have direct access communicate with the rest of the Internet through one of those that do.

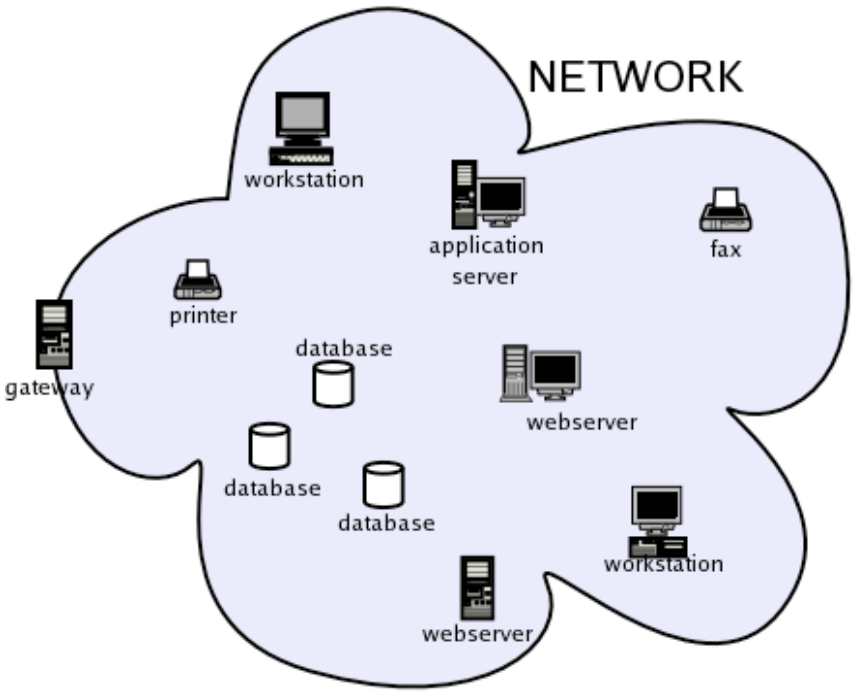

Figure 1.1-1: Resources on a network. Any device outside the cloud is considered a part of the Internet.

More recently network users are witnessing a new class of private networks called secure private networks or *Virtual Private Networks* (VPNs) [VPN04] [SBDG02] [MSFT00] [Fra01]. Apart from using access control, the gateways to these networks provide security to their users by encrypting information leaving or entering the network. This security allows many devices and network users to become a part of a *simulated* (or virtual) private network. These networks are 'virtual' or 'simulated' because the resources on these networks may not be within the same LAN yet, like regular private networks, the rest of the Internet may access these resources only through a gateway. Similarly, devices and users that are part of the network must access the Internet through a gateway. Because the various devices and network users of a VPN may not be on the

same network, VPNs achieve privacy by using encryption. Even though the Internet (generally consider public and so insecure) may exist between VPN users and resources, the VPN gateway encrypts all network traffic that must cross the Internet. A VPN gateway, thus, gives a network user *network presence*. This means that it creates the illusion that the user's machine is a part of some private LAN. Chapter 2 gives more details on the technology that allows VPNs to operate.

Access control is not only restricted to network access; devices on a network, and even the applications running on network devices may also enforce their own access control. This can present a number of access control boundaries or *network access rings*. Figure 1.1-2 below demonstrates how these rings interact. Each area within a ring represents a network entity: for example the area in ring A may be a private network, the area in ring B may be a machine running a network application (typically called a *server machine*), and the area within ring C is the application itself.

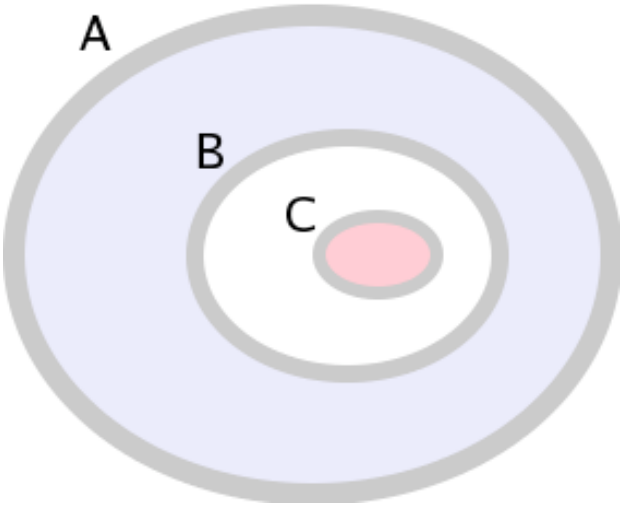

Figure 1.1-2: Network access rings

Each ring represents some access control policy that a network user must satisfy to gain access to the region within it (but not necessarily beyond an internal ring). In large networks, the manager of the network's gateway (called the *administrator*) - and thus the overseer of its access control policies - may not be the manager of the servers on the network. Because of this, access control policies are not typically *synchronized*. By this, I mean that the ability of a network user to satisfy the access control policy of ring A does not guarantee that she would be able to satisfy that of ring B as well. A sequence of rings as described above creates a hierarchy of access control that a network user must satisfy if she wishes to use an application in the region within ring C. The differentiation of access control by the network gateway, individual server machines on the network, and even applications on these machines, allows network resource administrators the freedom to decide who can or cannot use their resources. For remote users, however, these rings can be an impediment: because a remote network user may not be using the same host each time she connects to the network, she is now faced with ensuring that the host she uses at any one time can understand all the protocols and has all the credentials needed for each ring of access control.

Consider this scenario of a network user, Alice, and her colleague at work, Bob. Bob is the administrator for some server machine on the work LAN. Alice would like to be able to access some files on this machine, so Bob allows her to pick a username and password, and installs a Kerberos [NT94] [KN93] client application on her desktop at work. Kerberos is an authentication protocol that applications may use for access control. Kerberos requires that a network user's machine have a *client* application to present the

user's credentials – in this case the credentials are a username and password pair[2]. Alice can now access these files from her desktop at work because Bob has *authorized* her to do so. However, when she gets home later on and decides to continue the work she started during the day she realizes that Bob's Kerberos server would no longer allow her to retrieve her files. Naturally, her desktop at home lacks the Kerberos client, and so she realizes that her ability to access her files on Bob's machine is not just a consequence of who she is nor of the password and username she alone knows – it is also dependent on whether the machine she chooses to work from has the right applications installed.

In some cases, administrators may not want users to access a resource from just any host: within their own LANs these administrators may have some control over the *state* of each machine (whether the machine has a keystroke logger[3] installed for instance!) but cannot possibly know that of any remote machine. Bob could not have had such a measure in mind anyway because if Alice still wanted access from home she would 'only' have to find and install the Kerberos client on her home computer. I quote 'only' because the difficulty of such a procedure depends on the availability of the software and the ease of installation. For instance, even where the client support for a protocol (such as Kerberos' client applications) can be downloaded at no cost to a network user, installing the application might be quite an inconvenience for the average computer user.

Sometimes, it is *not* the case that administrators wish to restrict network users to computers with particular applications and credentials, or to specific locations. Instead,

[2] The Kerberos client does not actually present the username and password for authentication with the server; it uses them to obtain a digital 'ticket' that becomes the credential for authentication instead.
[3] A keystroke Logger is the generic name given to a program that can record characters a user types in some other application. Malicious users usually plant such programs onto machines with the intention of recording sensitive information like usernames and passwords.

some of these restrictions occur because access control policies depend on varying protocols, all of which require some software support that not every computer might have.

The following further illustrate current problems with access control. In most of these examples an administrator has granted a user access (the user is 'authorized') but the user is hindered by policy requirements. In others, access control policies simply do not capture the range of things a user would like to be able to do with their privileges:

1. Alice belongs to a team of consultants on the second floor of her office building. All of her team members keep their work files on the same file system server. Bob, the system administrator for their server, knows that the second floor computers all have IP addresses within a particular IP range. Each team member may access their files only after successfully supplying her specific username and password. Bob is an experienced administrator and understands that malicious users can more easily compromise usernames and passwords than they can many other authentication schemes. To improve the security of the server, he adds a *firewall*[4] that ensures that only connections originating from machines with one of the second floor IP addresses will be allowed to further communicate with the server. This is a typical IP-based (firewall) access control scheme for a device (in this case the server machine). While this provides some amount of security, it is really an inconvenience for the consulting team that would love to be able to work

[4] A 'firewall' is a set of access control rules that restrict access to a device based on information about a user's network traffic. Firewalls differ from other access control schemes in that they do not require the user to use an authentication procedure or protocol. They are typically used for preliminary access control only as a first line of defense from unwanted access.

from home. Even though Alice can get onto her work VPN she cannot access her team's server, which is really the only reason she wants to get onto her office's network in the first place.

2. Alice and her colleagues can check their profiles (personal information accounts) online. Charlie is the administrator for the web server that houses the profiles web site. Charlie understands that Alice and the other consultants may need to access their profiles outside work so he does not use IP-based firewalls. At the same time he also wants these profiles to be as secure as possible so would rather not use a username/password scheme. Instead, he sets up SSL authentication on his web server, requiring that the consultants' web browsers present a certificate, and prove knowledge of an associated private key for authorization. This is a PKI[5] (client-side SSL) access control approach for an application (in this case the web server). This approach is probably more flexible than Bob's but is an inconvenience to Alice who uses Internet Explorer 5.2 (the latest version of IE on Macintosh) on her Mac, which doesn't support client-side SSL authentication. What is more, it requires that the consultants have their private key and certificate installed on every browser they wish to use for accessing the web site. Again in this scenario, having access to their company's VPN does not buy the team much unless they meet the web server 's access control requirements.
3. Alice's team is currently dealing with a client, New Tech Co. New Tech has a VPN and has granted the team temporary access to its network. New Tech is using VPN access control for their network. Unfortunately New Tech's *VPN*

[5] PKI stands for Private Key Infrastructure. It is a term used for authentication based on public/private key pairs, certificates, and authorizing agents that issues these certificates.

concentrator uses the SSL[6] protocol for encryption. Alice's machine at home only has an *IPSec* VPN client because her workplace has an IPSec VPN and that is all she is used to. Alice's colleague, Denise, must deal with a different inconvenience: Unlike Alice, the VPN client on Denise's computer at home supports both SSL and IPSec protocols. However, Denise's VPN client can connect to only one VPN at any given time. This is an inconvenience for her because she would like to use some applications on her work VPN while retrieving files from New Tech's VPN.

4. Consider Eugene, a student at college. Eugene is going to be away in a foreign country next term, and he will not have frequent access to email. At the same time, he is expecting replies from potential employers and so would like to monitor his college's email account. What Eugene would like to do is have Felix, his best friend, monitor his email and call him up if he receives word from a potential employer. Now Eugene would really like Felix to be able to monitor the account until he gets back, and no longer than that. How does he do this without telling Felix his account password? Like most average web users, Eugene uses the same password for so many other accounts and he really wouldn't like Felix to be able to access those as well. Like most web services, the college's email access control policy has no notion of temporary or guest access.

[6] Please see Chapter 2 for more details about both the SSL and IPSec protocols

In all of these examples[7] the problems that the users face are not unsolvable: Eugene could resolve to practice better password habits, and change his other accounts to use different passwords, Denise and Alice could find and install another VPN client that handled both IPSec and SSL protocols and could simultaneously connect to multiple VPNs. Alice could get a different browser, for her Macintosh, one that supports client-side SSL. Charlie could issue portable USB tokens that carry the consultants PKI credentials to make accessing the web site more flexible. The administrator for the workplace's VPN could set up the DHCP server to issue the consultants' IP addresses within the range of those for machines on the second floor.

However, most of these solutions can be an inconvenience to the parties involved: How many accounts must Eugene fix-up and how many 'strong'[8] passwords can he remember? How comfortable are Denise and Alice with installing applications and how difficult would it be to install the client they need? How flexible is Alice at using another browser, other than Internet Explorer? What would it cost Charlie to get those portable tokens for the consultants and what learning curve must the consultants go through to become comfortable with these devices?

Neither do most of the solutions scale very well: Eugene may get more accounts in the future, just how many can he keep with unique passwords and will he ever have to allow some other friend to access one of these in the future? What happens when VPN technology requires some other protocol other than IPSec and SSL and where does the introduction of new VPN protocols end? Which platforms or operating systems support

[7] Any technologies that might be unfamiliar to the reader are further explained in Chapter 2 and the glossary section of this paper.
[8] The definition for a 'strong' password is rather ambiguous, but generally passwords that have many characters (8 or more), and include numbers and non-alphanumeric symbols are considered 'stronger' than those that do not.

Charlie's tokens and do all of these have drivers already installed on them? What happens if another administrator, Gina, decides to use IP-based filtering, and what if Alice happens to be in the group that uses Gina's server machine as well? From which IP address range would the VPN administrator assign Alice's address, the second floor's or that for the team that uses Gina's server machine?

In this paper I introduce an access control solution, called the Access Control Service, or ACS for short. The ACS provides a VPN service, so like other VPN gateways it provides network presence for remote users. The ACS also manages user account information and so is aware of what privileges each user has. The ACS provides a centralized framework for resource administrators to autonomously control access to their resources. The ACS is also able to facilitate remote-user access to resources on the VPN by establishing connections to these resources on behalf of VPN users. The ACS is, therefore, a mediator between a remote network user, and the resources that she is authorized to access. Because the ACS mediates on behalf of the remote user, a remote user will be able to access network resources for which she has permission irrespective of the particular host she might be using at any time.

Also because the ACS is an intermediary, it becomes the ACS's responsibility to be able to interoperate between various protocol versions and machine capabilities; each resource administrator only has to ensure that the ACS can satisfy the resource's access policies.

Some of the work of the Greenpass team at Dartmouth has shown that delegated access to a network can be extended to VPNs as well [Goff04]. Unfortunately, in the light of the problems highlighted above, and in the absence of any delegated access control

schemes for individual network resources, delegated access to VPNs may not be of much use. The previous work on delegated VPN access, however, points out that VPN users may be authorized either by direct permission from an administrator or by delegated permission from another authorized user. This suggests that resources on a VPN must be able to accommodate both types of users if delegation should be seamless across the network. The ACS framework provides a solution for this problem: the ACS can incorporate Greenpass tools for determining delegated users' privileges, establish connections with resources on the VPN, and allow authorized delegated users to access these resource via the established connections. Since the ACS is an intermediary between delegated users and these resources, administrators may not have to make major to their access control systems in order to accommodate decentralized delegated access.

Summary

While VPN technology offers authorized users network presence, it does not cater for the varying access policies on the network. This means that being able to obtain network presence via VPN access control may not benefit a remote user much because the target resources may require further authentication that her remote host may not be able to satisfy. These restrictions also mean that changes to the access policy of a resource on a private network may suddenly prevent an authorized remote user from accessing this resource. Thus user access no longer becomes based on who the user is but on what the user's host has or can do at any given time. Access protocols are typically meant to establish a user's identity and/or to ascertain a user's privileges. However,

where a user's host does not correctly support an access protocol, these protocols end up restricting access. Consequently, such restrictions present an inconvenience to both users and resource administrators. At the same time simply changing the access control policies for these resources cannot be the solution because: (1) network users are necessarily restricted by the capabilities and possessions of their host machines. (2) access control policies are designed with more than just authorized VPN users in mind. They are therefore necessary for providing security to these resources. A solution lies in a mediating service that understands the privileges of each remote user and the access control policies of their target resources on the VPN. By proxying authorized access, the ACS can alleviate the inconveniences caused by network access rings as well as provide a platform for introducing other network-wide technologies including decentralized delegated access.

1.2 Motivation

The motivation for this work arises from a rather interesting problem that exists on the Dartmouth College network, but similar cases exist on many other networks that use IP-based access control schemes for *secured server machines*. A secured server machine is a computer that runs a number of services and is secured by some access control policies; it only allows authorized users to make connections to the services it runs. At Dartmouth, there exist a number of server machines that enforce access control policies requiring that an authorized user possess an IP address within a particular

address range. Thus, these machines perform preliminary authentication by IP-filtering (firewalls), i.e. the access control policy allows only users with specific IP addresses to connect to the server machine (or, alternatively, users with specific IP addresses are disallowed access). The problem with IP-filtering is that it is not the most sophisticated scheme for securing a computer; a malicious user need only spoof an IP address to be considered authorized to make a connection. Alternatively, malicious users can thwart the policy by employing a number of *packet* fragmentation attacks [ZRT95]. A network packet is a unit of network data. Because these computers discriminate purely based on the IP address of the connection requestor, a number of problems may arise:

1. Most host machines on the network use dynamically assigned IP addresses so in order for the host to connect to one of these server machines it must be assigned the appropriate IP address. A problem arises when a particular host desires to access more than one of these machines with non-overlapping IP address ranges.
2. When a user has permission to access two of these machines, system administrators may use overlapping IP address pools so that the overlaps represent users with access to both machines. This obviously doesn't scale well when a user is authorized to access three or more of these server machines.
3. Network Administrators may then decide to split the local address space into ranges corresponding to privilege classes, where each privilege class represents a permutation of the permissions a user has. Here problems include scaling the IP range to accommodate changes in the number of users in the corresponding class,

and adding or removing server machines that work on this access protocol scheme.

4. Alternatively, the administrator may create access control lists (ACLs) for each server, listing some fixed IP addresses to which each server may grant (or deny) access. This approach may reintroduce the problem of multiple permissions identified in (1) if servers do not share IP addresses. Consequently, a number of these servers may have overlapping IP addresses that represent the intersection of their users. Again, this approach makes modifying user privileges rather cumbersome because an administrator must modify the ACLs of each server for which the user has access permissions.
5. Access control based purely on IP addresses does not work well with temporary access permissions. It usually involves the administrator modifying ACLs for a number of computers, or adding new IP addresses to secure server machines, and then undoing the process when the privileges expire.

At the same time filtering connections by IP addresses has advantages that make this approach a welcome choice even with the existence of more sophisticated ones. Amongst these, perhaps one of the more motivating factors is that packet filtering by IP addresses is done in the computer's *kernel* (the part of the operating system that directly interacts with hardware) so can be executed very quickly. Other approaches require user authentication that may require userland applications (applications that run on top of the operating system) and protocols that are considerably slower. Thus many of these server machines only use IP-filtering for preliminary authentication; further application-level

authentication might be carried out by individual services after the machine accepts a network packet from a user. Thus IP-filtering serves to reduce the number of users a server must authenticate in a slower way.

Nonetheless, because of its associated problems outlined above, and the vulnerabilities mentioned earlier, the problems of maintaining access control by IP-filtering can very quickly outweigh its use as a security measure in larger networks. For many networks, including Dartmouth's, administrators have not considered eliminating IP-based access control or adopting more sophisticated schemes for a number of reasons:

1. The network does not yet have too many of these server machines, so maintenances has not yet become unbearable. On the Dartmouth Network, there are almost 20 such computers though administrators admit that this number is growing.
2. The network is still not so big that there are too many overlapping users or a lack of IP address ranges to specify new overlaps.
3. Guest access has not yet become an issue or has never really been considered.

While these are valid arguments they all assume that the network will not expand or require more sophisticated access control (like delegated access).

The Dartmouth network employs a VPN concentrator as a gateway to most of these server computers. At the time of researching the above problems, I was also researching VPN usage on the network and its integration with delegation tools [Goff04]. An approach to mitigating the problem of maintaining IP-based policies for secured

server machines is presented in the ACS mentioned earlier. The ACS makes use of the authentication provided by VPN technology, as well as network address translation functionality to ensure that authorized users always meet the IP policy of the secure server machine. All this functionality is configurable so can be modified to reflect the correct freedoms and restrictions that remote users would experience had they been locally present on the network.

1.3 Goals of the ACS project

The goals of the Access Control Service are:

1. To enhance VPN user access by reducing undesired access control restrictions to network resources. These restrictions are those due to protocols that a remote host may not be able to handle or policies the remote host cannot meet even though the remote user has access privileges.

2. To facilitate maintenance of the access control systems for several resources, perhaps under the supervision of several administrators, by providing a centralized framework defining the privileges of users and user-groups in accessing resources and resource-groups. Sine the ACS is an intermediary in the network-access process administrators can modify user and user-group privileges

on the ACS to reflect permission changes without having to change individual access control policies on multiple network resources.

3. To provide a centralized framework for introducing network-wide access control technologies and schemes such as decentralized delegation of resource access.

1.4 Organization, Style and Formatting of this document

Throughout this document the word 'I' is used in contexts where I am referring to work I did 'on my own.' I quote this phrase because while I coded up, put components together and wrote this document, I acknowledge the very helpful suggestions and input of my advisor and others (see **Acknowledgements** section). I use 'we' when I mean to engage the reader in discussion as in "we have previously identified a number of issues…" Where I use 'we' in this context, I will make it as clear as possible that it refers to the reader and I.

I will refer the reader to sections within the **Background** Chapter (Chapter 2) each time I make reference to some technology that I believe the reader might not be acquainted with. When I introduce a term for the first time I will *italicize* it. I will endeavor to further explain the term almost immediately after its introduction. If I do not immediately provide a definition for one of these terms, then a definition is included in the **Glossary** section of this paper. I may also place a term in quotes if I believe the term

is familiar to the average reader but might require further explanation in the particular context. If I have to make an emphasis, I will underline a phrase to emphasize it.

I generally use the term 'server machine' to refer to the physical device that accepts connections and provides a number of services (like the ACS). I use the term 'server' to refer to the specific service application on a machine. I will use the term 'client' in two ways: (1) the application on a host that can understand a protocol required for connecting to a server. (2) A remote user who connects to a network server. I have already used client in the first form in this chapter. The context will make either use relatively unambiguous to the reader.

I use 'public,' 'insecure,' and 'less trusted' synonymously when referring to networks. The underlying idea in all three cases is that any device on the Internet can intercept traffic on such a network so there is no confidentiality. Second, in the absence of encryption, the integrity of the data through this network cannot be guaranteed. Likewise, I use 'private' and 'trusted' synonymously; devices on the Internet are less likely to intercept traffic on such networks. Devices within these networks may, however, be able to intercept the traffic.

The paper is divided into chapters based on the nature of the information I will be presenting:

Chapter 2 highlights the main technologies employed by the ACS project. I strongly encourage the reader to at least skim this chapter before reading about the project itself.

Chapter 3 presents the model for the ACS as well as detailing the design and implementation of my prototype for the framework.

Chapter 4 discusses some of my thoughts on the current implementation with regard to security, usability and efficiency. These discussions also touch on my ideas for improving the prototype as well as the overall model.

Chapter 5 presents other research pertaining to network access control management and delegated access for network users.

Chapter 6 is the conclusion to the work presented in this paper.

1.5 Related Documents

This project was built upon Nick Goffee's experiment on delegated access and VPNs [Goff04]. His experiment was carried out as part of a larger thesis related to the Greenpass Project at Dartmouth. The Greenpass team has submitted a number of papers on this project [SGK+04a] [SGK+04b], and there have been two masters' theses [Goff04] [Kim04] on the development of the RADIUS server and Client Tools components, as well as an undergraduate thesis [Pow04]. More information about Greenpass is provided in Chapter 2.

Jacco De Leeuw [DeL02] has an online document about setting up Linux VPN servers using open source tools.

Chapter 2

Background

2.1 Virtual Private Networks

2.1.1 Definitions

VPNs are private networks made by one or more connections over a public network infrastructure (the Internet). VPNs are typically accessible by means of one or more entry points called *Points of Presence* (POP). Figure 2.1.1-1 below shows a typical VPN setup for an organization with access capabilities for a regional branch, a remote business client, a partner and a remote employee.

In most organizations, a *concentrator* authenticates users before they gain VPN access. In network terms, a concentrator is a device that "allows a number of stations to be connected to a LAN" [Wob97]. In the case of a VPN concentrator, this 'LAN' is the VPN, and the 'stations' are remote network devices. The term 'VPN server' defines software on a machine (a gateway) that accepts connections from a remote host to enable the host to access other services on a private network. VPN servers traditionally run on PC and Mac server machines, Solaris workstations etc. As the machines running the server are not specialized for routing traffic, or handling larger volumes of connections, VPN servers are typically used on gateways to smaller subnets rather than the entire networks. Traditionally, 'VPN concentrators' refer to devices specialized for VPN access control; they may audit and monitor connections more efficiently and provide hardware support for encryption and increased connection throughput.

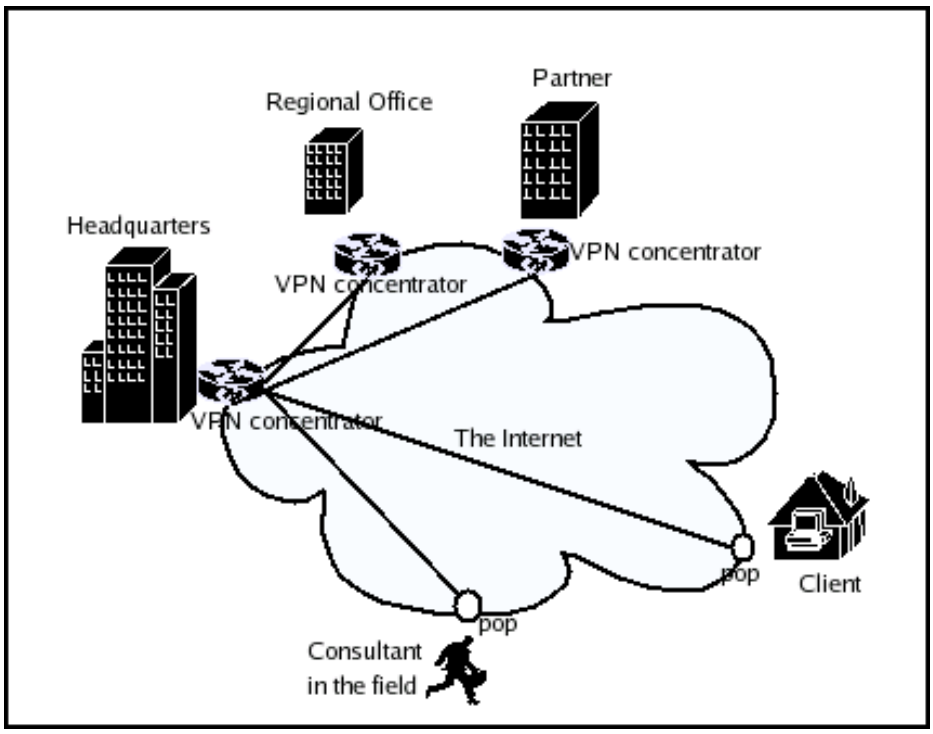

Figure 2.1.1-1: A VPN setup for a corporation, its employees, partners and clients.

VPN servers can be 'inward-facing', 'outward-facing', or both. Outward-facing servers are the public interface to a virtual private network; they receive incoming connections and forward data from remote machines to devices on the VPN and to the rest of the Internet. Inward-facing servers are the gateways for devices within the VPN; they forward packets from within the VPN to the rest of the Internet and tunnel packets destined for other servers.

VPN connections are usually referred to by the traffic that is encrypted between the two hosts on either ends of the connection. In the most basic form, a VPN connection encrypts only those packets destined for one of the two hosts. This configuration is called a *host-to-host* VPN. Second, a client may connect to a server for access to only the subnet behind the server's host (the gateway). In this configuration, called a *host-to-subnet* VPN,

the client's traffic to the rest of the world remains unencrypted, only packets destined for hosts on the remote subnet get encrypted. The third type of configuration is the *host-to-anywhere* VPN. Under this setup, all the client's network traffic gets encrypted and sent to the VPN server, even those packets that are destined for hosts outside the VPN subnet. The VPN server will typically forward these packets to the rest of the Internet.

In all cases, VPN connections are considered to offer *complete site-to-site encryption* [Hos04]. 'Complete site-to-site encryption' means that all data between the VPN client's host and the VPN server's host is encrypted.

Closely related to VPNs are 'virtual LANs' (VLANs). VLANs are network devices that are clustered based on logic rather than geographical location. These devices maintain connections among themselves using software configurations. This allows a device to remain a part of the VLAN even after its physical location is changed. The fundamental difference between a VLAN and a VPN is that the definition of a VLAN does not consider security or privacy.

In networking, 'tunneling' refers to "the practice of encapsulating a message from one protocol in another, and using the facilities of the second protocol to traverse some number of network hops" [CBR03]. The VPN Consortium's white paper [VPN04] defines three VPN tunneling technologies; trusted VPNs, secure VPNs, and hybrid VPNs.

- Trusted VPN tunneling is equivalent to private leased-lines technology; a corporation that employs a trusted VPN makes use of a network provider's private network nodes, such as switches and routers, with the agreement that the provider ensures the integrity and privacy of data through the network.

- Secure VPNs make use of publicly-available circuitry and so they must employ encryption to achieve the same effect as trusted VPNs.
- Hybrid VPNs, as the name suggests, make use of both private network circuits and encryption.

Tunneling (with or without encryption) can be achieved by a number of protocols. This section will cover four of these protocols: IPSec, L2TP, PPTP and SSL.

2.1.2 IPSec

The Internet Protocol Security (RFC 2401-2411, 2451 Standards Track) is an Internet Draft Standard that was developed by the Internet Engineering Task Force (IETF) to ensure secure transfer of information on a public IP network [Fra01]. IPSec is a per-packet security implementation that consists of two independent protocols: The Authentication Header (AH) [KA98a], and the Encapsulation Security Payload (ESP) [KA98b]. Both protocols describe the IP header extensions for sending and receiving protected datagrams. ESP provides privacy through encryption (and optionally integrity) while AH provides packet integrity and authentication [SBDG02]. The IPSec protocol also specifies the Internet Key Exchange (IKE) protocol [HC98], which is used for authenticating, and for establishing session keys between two sites before any other data is transmitted.

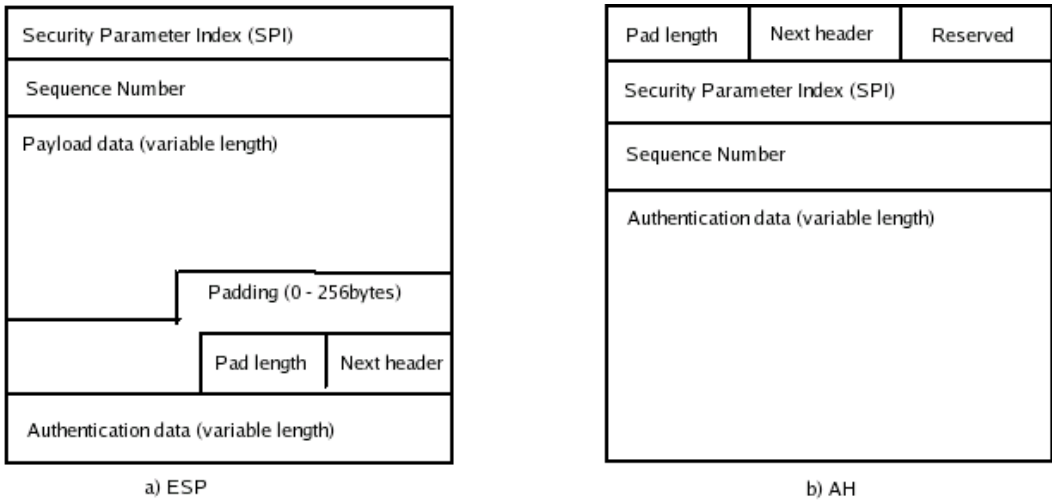

Figure 2.1.2-1: IPSec information part of IP packet

IPSec specifies two modes of applying protection:

1. Transport mode adds IPSec information (ESP or AH) between the IP header and the remainder of the packet.

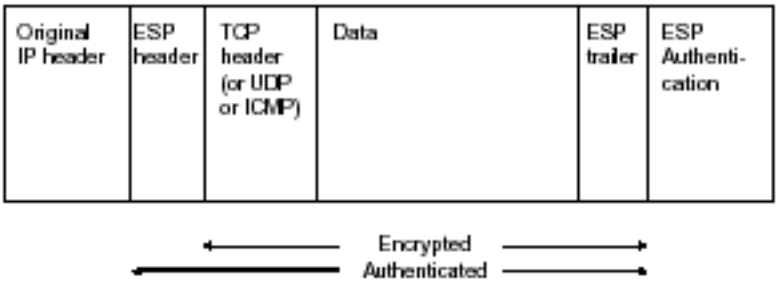

Figure 2.1.2-3: IPSec using ESP in Transport Mode

2. Tunnel mode adds a new IP header to the original IP packet, and keeps the IPSec information outside.

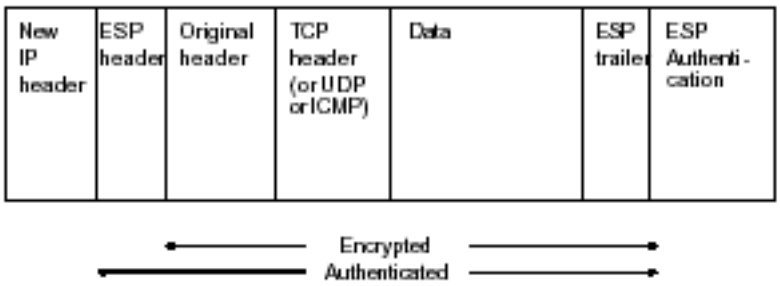

Figure 2.1.2-2: IPSec using ESP in Tunnel Mode

A security association (SA) is a relationship between two or more entities that defines how each entity will use security parameters and services to communicate securely [KPS02]. An IPSec SA is a cryptographic key and other information including the identity of the host on the other end of a connection, the sequence number currently being used, and the 'cryptographic services' being used [KPS02]. The 'cryptographic services' tell each end such information as whether to use integrity only (AH), or encryption and integrity (ESP / AH + ESP), and which cryptographic algorithms to use. The purpose of IKE is to establish an SA between two parties. SA information on each IPSec connections is stored in an SA database and indexed by a security parameter index (SPI) that is included in the header of each IPSec packet (see figure A.1) and is unique for each connection to a given IPSec server.

2.1.3 PPTP

The Point-to-Point Tunneling Protocol (RFC 2637) is a layer-2 protocol built on *Point-to-Point protocol* (PPP) and TCP/IP. PPP is a mechanism for creating and running the Internet Protocol (IP) and other network protocols over a serial link – be that a direct

serial connection (using a modem cable), over a telnet established link, or a link made using modems and telephone lines (or digital lines) [LDOC]. PPP allows authentication, privacy and data compression, and is commonly used in dial-up Internet connections via modem and phone lines. PPTP tunnels a PPP session through IP connections and so benefits from the security provided by PPP while using IP infrastructure. PPTP thus allows the transmission of packets that do not support IPv4 or IPv6 addressing standards. The vendor consortium responsible for developing this protocol includes Microsoft, US Robotics, and 3Com but the Microsoft implementation appears to be most widely used. PPTP uses a variety of protocols for authentication and encryption. Authentication protocols include Password Authentication Protection (PAP), Challenge Handshake Authentication Protocol (CHAP), and MS-CHAP (A Microsoft adaptation of the original CHAP). Encryption algorithms typically employed include RSA, RC4, and DES.

2.1.4 L2TP

Cisco Systems and Microsoft developed the Layer-2 Tunneling Protocol (RFC 2661) to combine the advantages of PPTP and an older Cisco layer-2 tunneling protocol called Layer 2 Forwarding (L2F). L2TP enables the transmission of PPP datagrams over IP, X.25, *Frame Relay* and *ATM* networks, and allows the establishment of multiple tunnels between the same endpoints [Fra01].

A typical L2TP implementation consists of two services; the L2TP Access Concentrator (LAC) and the L2TP Network Sever (LNS), and two tunneling modes; Voluntary or Compulsory. The LNS is the endpoint of an L2TP connection and the LAC

is the intermediate service that routes packets between the LNS and a remote host at the other end of the connection. The LAC is able to receive non-IP connections and tunnels PPP datagrams to the LNS for authentication. The LNS receives L2TP connections from the LAC, processes PPP datagrams, uses PPP authentication to validate users, and assigns IP addresses. In compulsory tunneling mode, a client dials into an Internet Service Provider using a PPP connection. The ISP partially authenticates the client to establish which account to open, and then the LNS validates the client and assigns an IP address. Voluntary mode is very similar to compulsory mode only that the client already has a pre-establish connection to an ISP so no partial authentication is required.

L2TP uses Network Control Protocol (NCP) for assigning IP addresses and authenticates using PPP authentication schemes (CHAP or PAP) in order to control network access. L2TP is primarily concerned with user authentication, and the confidentiality and integrity of information between L2TP endpoints. L2TP does not provide replay protection.

2.1.5 Comparisons

Authentication provided by L2TP and PPTP is user-based and occurs during tunnel establishment only. Neither provides authentication of a packet's origin thus both lack replay protection. IPSec authentication is host- rather than user-based, because IPSec only identifies packets by IP addresses after a tunnel is established[9] [KPS02]. IPSec provides replay protection through origin authentication and by maintaining sequence numbers of each packet in an SA database. However, IPSec does not support dynamic

[9] IPSec servers can potentially handle multiple connections between the same end points because they identify connections by SPI rather than IP addresses. However, most IPSec clients do not allow this.

configuration of parameters after the tunnel is formed [RS02] because once the SA is established each end of the connection must remain true to initial agreement during the IKE phase otherwise one end point will reject the others packets.

IPSec/L2TP (pronounced IPSec over L2TP) is typically used for VPN implementations. IPSec over L2TP combines the replay protection advantage of IPSec with the user-based authentication and routing ability of L2TP. This allows roadwarriors (mobile devices with non-fixed IP addresses), which may not have IP addresses prior to establishing a connection, to use IPSec/L2TP.

2.1.6 SSL

Netscape first introduced the Secure Socket Layer (SSL) protocol for securely sending messages over the Hypertext Transfer Protocol (HTTP). HTTP is the native protocol of the World Wide Web and so SSL was initially intended specifically for browser use. The protocol has existed for several years now (Netscape released the first official version, SSLv2 in 1994) and has developed in security and feature. Currently, the IETF's standard for SSL is the *Transport Layer Security* (TLS) protocol, which is a modified version of SSLv3.

2.1.7 SSL VPNs

Unlike the previous protocols, SSL was not designed with layer 2 and 3 encryption in mind, but rather transport-layer encryption. There are four methods by

which SSL VPNs may provide access: *proxying*, *application translation*, *port forwarding* and *network extension*. Some implementations may use more than one of these:

I. Proxying

An application proxy is a service that sits between a client and remote server and acts as an intermediary between the two for a particular application. The proxy usually receives data from one end (client or server), may rewrite this data in some way (including encrypting or decrypting it), and then forward the data on to the other end. These proxies are commonly called *Application Level Gateways* (ALGs). A web proxy is an ALG that handles HTTP requests between a client and server. ALGs typically proxy a small subset of applications because they are protocol-dependent. Consequently, proxying cannot offer complete site-to-site encryption that is desired in a VPN.

II. Application Translation

Protocols like FTP can be adapted to other protocols like HTTP (used by web browsers). Proxy servers can do this translation on behalf of clients, in order to handle requests to servers. However, more complicated protocols, like the one employed by Windows File Sharing, are less readily translatable. This mechanism, like proxying, is protocol-specific and cannot offer complete site-to-site encryption.

III. Port Forwarding

Port forwarding is a type of network address translation typically used by servers to balance load. A host machine may receive a packet on one port and forward it to a port of another machine.

IV. Network Extension

This is the use of a tunnel to join two hosts at the network layer. Such tunnels do not discriminate based on application protocols because they built below the application layer.

SSL VPNs that use network extension can successfully encrypt all data between two hosts and so are regarded as 'true'[10] VPNs [Hos04]. I will use OpenVPN, considered a 'true' SSL VPN [Hos04], to explain how SSL VPNs work:

OpenVPN [Yon04] is an open source SSL VPN implementation created by James Yonan. OpenVPN uses the OpenSSL libraries and is available on all the common computing platforms (Windows XP, Mac OS X, and Linux). OpenVPN uses virtual interfaces, much like PPP does. It supports two virtual interfaces, a *tun* interface, which is a tunnel interface just like PPP's, and a *tap* interface. The *tun* interface routes IP packets, while the *tap* interface routes Ethernets packets. Each interface has a *virtual device driver*. The device driver is a user-land application that processes packets sent to the interface. Before the virtual interfaces are created, OpenVPN will perform a SSL handshake with the remote host. This handshake is a client-SSL authentication handshake so both ends of the tunnel get authenticated. Once the connection is established, packets can be routed to the remote host. Because the *tun* and *tap* interfaces are network

[10] SSL VPNs are considered 'true' VPNS if they provide complete site-to-site encryption

interfaces the host operating system sends them entire IP (or Ethernet packets), not just application layer data as it would a regular process doing SSL encryption. The driver receives these packets and encrypts them employing algorithms from the OpenSSL library (*DES*, *3DES*, *AES*, etc). It then re-encapsulates the new packet in a UDP packet, addresses this packet to the remote host, and sends it onto the network.

Unlike IPSec servers, tunneling is done over UDP rather than plain IP. The reason for this is that OpenVPN (and any other SSL VPN implementation for that matter) does not have kernel support, so it must listen on a specific port to receive VPN packets. Tunneling is done over UDP, as opposed to TCP, because the encrypted (tunneled) packet may already be a TCP packet; tunneling TCP over TCP can cause network slow-downs because each TCP layer must handle resending requests when a packet is lost, and each layer must ensure that packets come in order.

SSL VPNs have a number of advantages over IPSec (or IPSec/L2TP) VPNs. Perhaps most notable amongst these is that they do not require any modification to a host machine's kernel stack. This advantage appears to make them much easier to install and configure, and allows them to operate more easily between different platforms [Yon04]. As a disadvantage, SSL VPNs are less common and less widely used by organizations. Due to this SSL VPNs have a lot fewer router and hardware support than their IPSec counterparts. One reason the public might have been so slow to pick up SSL VPN technology is that there is a common misconception that SSL only works at the application layer so cannot provide more than per application security [Hos04].

2.2 Network Address Translation

The necessity of NAT arises from the inability of IPv4 to provide enough addresses for the Internet [CBR03]. NAT tries to circumvent this limitation by constructing a many-to-one mapping of IP addresses within a local network so that machines on this network appear to be sharing the same IP address on the Internet. The way a *NAT box* (the device doing Network Address Translation) achieves this varies from implementation to implementation, but the common approach is to assign all outgoing traffic (packets leaving the local network) the IP address of the NAT box, and multiplex incoming traffic (packets entering the local network) by mapping destination ports to machines on the local network.

2.2.1 Problems associated with NAT

Most of the problems with NAT arise from the fact that some application protocols (e.g. FTP) use IP addresses in remote commands at the application layer. These IP-address references are not readily obvious to a NAT box so may be left untranslated and cause problems to the applications. NAT implementations must, therefore, handle FTP packets specially in order to circumvent this problem. This introduces a system where NAT boxes have to handle packets on a protocol-by-protocol basis, which is not very scalable.

Second, because the source IP addresses of outgoing packets are modified, hosts behind a NAT box cannot readily act as servers; remote clients would not have a static public address to which to connect. This particular problem also affects applications that ignore server-client models in favor of 'peer-to-peer' connection (e.g. voice over IP, X-windows) because such applications require that either side be able to accept a connection (both sides of the connection are clients and servers at the same time).

Third, the process of translating IP headers introduces extra overhead to data packet flow on the Internet. Furthermore, translating addresses violates the standards of some security protocols such as IPSec; IPSec enforces packet integrity and under the AH protocol IPSec requires that the IP header remains unchanged throughout the packets transmission. NAT easily violates this standard because it requires that packets be modified at least at the header level [KPS02]. While there are some implementations of IPSec that permit NAT boxes to modify headers, such implementations introduce extra complexity to IPSec usage and still cannot accommodate some packet tampering that NAT requires. For instance, some NAT implementations handle FTP packets by tampering with application layer information. Application layer data falls within the payload of a typical IP packet and so is considered un-modifiable according to the IPSec protocols.

2.2.2 Practical applications of NAT

NAT has become almost crucial in solving certain problems of security and balancing load for servers and databases [Vep02]. A server may use NAT to hand off

packets destined for it to clone servers on other machines to deal with. This allows a server at a single IP address to balance client load. This scheme is used to handle high volumes of database queries. Additionally, NAT is used to provide a single gateway to multiple machines. Such a setup is called *masquerading* or 'many-to-one *natting*[11]'. The reasoning behind this configuration is that it is easier to secure one gateway to the Internet than multiple ones. Thus firewall rules applied to the single gateway provide security to all the masqueraded machines. Masquerading is also used to protect the identity and activity of machines on a network.

NAT is becoming more and more popular in establishments like airports and hotels. In such establishments, rather than register an entire subnet of public IP addresses, the establishment offers the patrons local addresses (like 10.0.0.1) and a NAT box masquerades outgoing traffic for them with a few public IP addresses. This setup is much more cost effective for these enterprises. The Internet Assigned Numbers Authority (IANA) has provided three IP address ranges for local network use [RMKGL96]. These address ranges are given in Table 2.2.2-1.

Start address	End address
10.0.0.0	10.255.255.255
172.16.0.0	172.31.255.255
192.168.0.0	192.168.255.255

Table 2.2.2-1 IANA assigned address spaces for local networks

[11] 'Nat' is sometimes used as a verb and means, "to do a network address translation."

2.3 Linux Packet Routing

2.3.1 Definitions

Packet routing allows a machine to discriminate between packets it receives or transmits. This permits a host machine to behave like a regular network router. In many cases, packet routing enables a host to also make other decisions about packets including modifying packet header information such as *time-to-live* and *type of service* fields (*mangling*), or changing source or/and destination addresses (*network address translation*). Packet routing can be said to provide three main functionalities in Linux:

1. Denying or permitting packets (whether those entering, or leaving the host). This type of discrimination allows a host to *firewall* traffic to and from it. A firewall is a set of rules a machine uses to make decisions on packets (whether to reject or accept), based on network layer information. This information is typically which port or address the packets originated from or to which port or address it is going. In Linux, tools such as *iptables* [NTFLTR] provide this functionality.
2. Deciding which address or network device to use for transmitting a packet. A network device can be a physical device, such as a *Network Interface Card* (NIC), or a virtual device, which is the way systems like Linux represent tunnels (e.g. PPP or IPSec). On a Linux machine, the *route* tool provides this functionality.
3. Changing some attribute about a packet entering or leaving the machine. This includes *mangling*[12] and network address translation, already discussed above.

[12] Mangling is the process of changing time-to-live and time-of-service information on a packet header.

Firewalls may make decisions on packets by inspecting the packet header. This is called *static packet filtering* [PCWEBa]. Another approach is to make decisions based on the packet's activity (and that of related packets) through the various interfaces of the host machine. This approach is called *dynamic packet filtering* or *stateful inspection* [PCWEBb]. Stateful inspection allows decision to be made on packets based on previously established or related connections. The *conntrack* module of iptables is responsible for this functionality.

2.3.2 Iptables

Linux provides packet routing through iptables on kernel versions later than 2.3 [Russ02]. Iptables is the successor of the userland tools *ipfwadm*, from Linux 2.0, and the more recent ipchains, for Linux 2.2. Both of these were based on BSD's *ipfw*. Iptables is a part of the Netfilter framework, which is a set of modules for the Linux kernel (these modules could optionally be built into the kernel). The Netfilter framework offers hooks into the kernel's packet processing mechanism. These hooks differentiate packets destined for *processes* within the host, those leaving the host, and those being forwarded to other network devices or back to the network. The framework also offers a series of 'chains' that mark the various stages a packet has reached in the kernel's packet-processing mechanism. Associated with these chains are two tables, nat and mangle. Nat is the Network Address Translation table and is defined for only the PREROUTING and POSTROUTING chains. Figure 2.3.2-1 below shows these chains and corresponding tables.

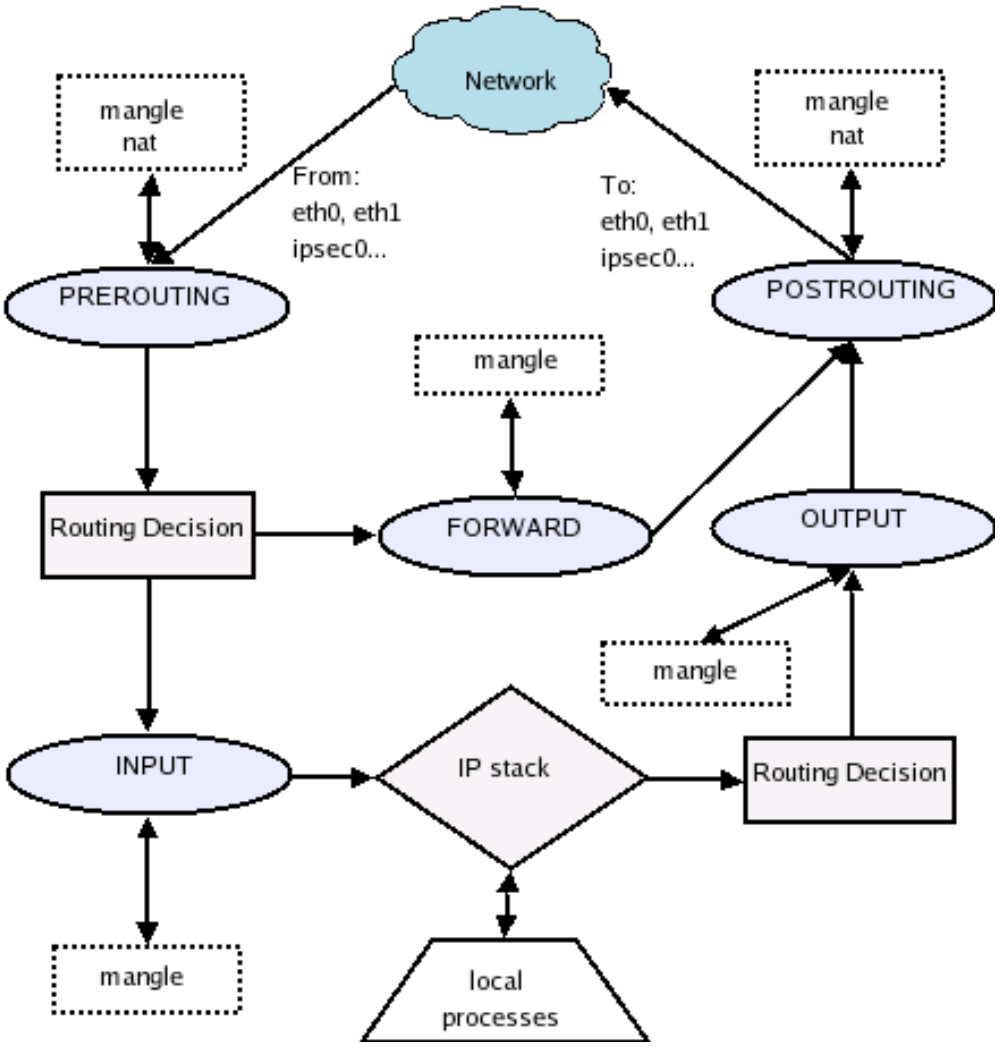

Figure 2.3.1-1: Iptables chains and tables

On each chain, the mangle table is visited before the nat table, if both exist. All rules are processed sequentially. Each chain can also be assigned a default target decision. A target, for most chains, is either ACCEPT or DROP. Other targets are QUEUE and LOG. The QUEUE target causes the kernel to queue a packet that reaches that rule. A process on the host machine that has a handle to the Netfilter queue may then read the queued packet. Netfilter provides a library, 'libipq,' through which applications may read and process queued packets. The LOG target uses *syslogd* to log packet

information using the kernel facility (i.e. as kernel messages). The target allows specification of a priority for these log messages. Iptables also allows the creation of user-defined tables. In fact, Linux' *Lokkit* [RHLa] uses this feature for creating firewalls.

Iptables provides a module, called 'conntrack', for tracking connection state. This module identifies the start of a TCP connection when a SYN packet is transmitted from one site to another. The connection is considered *established* when a reply SYN/ACK packet is transmitted to the site that initiated the connection. The connection is considered terminated (or closed) when a FIN/ACK – ACK communication is transmitted in both directions [And03]. A RST packet can also be used for terminating the connection. For a connection over UDP, the module considers the connection 'new' when the first UDP packet is transmitted from one site to the other. The state of the connection changes to 'established' when a UDP packet is observed in the other direction. The connection remains 'established' for a predefined amount of time and times-out (is considered 'closed') if no more activity is seen between the two sites.

The conntrack module can also handle ICMP: An echo request packet starts a new connection and an echo reply packet ends it. Most other echo packets will end the connection (e.g. 'net unreachable' or 'net prohibited' packets). ICMP connections also have a default time out (about 30 seconds). Conntrack will consider a connection 'related' if there is already an established connection between the two sites. Thus an ICMP packet sent by one site to another to confirm the state of a UDP connection will not generate a new connection under iptables. If conntrack does not know the packet protocol it uses a default behavior to track connection state. The default behavior is much like the UDP connection approach explained above.

Because chains, tables, and rules are traversed sequentially, complicated routing systems can easily slow down traffic on the machine [And03]. One way of speeding up rule-processing is to use sub-tables; rather than have all rules exist in one chain or table, packets can be categorized, and each category assigned a different sub-table, thus reducing the number of rules that have to be traversed for each packet. Categorizing packets can be made easier by marking packets that arrive from various interfaces. For example, packets arriving at a host via ipsec0 (ipsec0 is the virtual interface for an IPSec tunnel on Linux 2.4 using *KLIPS*) could be marked to distinguish them from packets arriving at other interfaces. Later, packets with this mark could be processed in a user-defined table so that other packets do no have to traverse rules intended for a VPN client.

2.4 The Greenpass Project

Dartmouth's Greenpass project seeks to enhance wireless access control to networks by introducing *decentralized delegated access* [Goff04]. Decentralized delegated access allows network access to be determined by a number of *principals* rather than one central authority. In *PKI* terminology, a principal is any authority that signs a certificate or vouches for a network user. In the Greenpass project, these principals are regular network users with delegation privileges. A *delegator* [Goff04] is such a privileged user. When a guest wishes to join a wireless network, the guest presents a certificate that is signed by a trusted *Certificate Authority* (any certificate issuing agent)

to the *Access Point* (AP)[13]. The AP forwards this certificate to a *RADIUS (Remote Authentication Dial In User Service) server* [Rig00] [RWC00] [RWRS00]. The RADIUS server decides if the AP should accept the guest's request and the AP responds to the guest accordingly. Guests without the right credentials are shunted to a restricted VLAN from which they can access only a single delegation website. Here, the guest must make a request to be granted access by a delegator that they know. Once approved by the delegator, the guest is given a token as proof of authorization. This token (a web cookie) is actually a *SDSI/SPKI* [EFL+99] certificate chain (hereafter referred to as SPKI certificates for simplicity) showing the relationship between the guest and their delegator. SDSI/SPKI is a certificate specification (much like X.509) that has built-in support for delegation in the form of certificate chains.

By restricting network access to permit only authorized guests, the project enhances the wireless *EAP*-TLS authentication. The *Extensible Authentication Protocol* (EAP) [BV98] is a standard that was originally created for authenticating PPP sessions. EAP provides a standard for encapsulating various forms of authentication handshakes including challenge-response schemes, password-based, Kerberos, and PKI-based methods (like TLS) [SGK+04b]. EAP is a protocol that forms part of the standard called *802.1x* for wireless control to Ethernet networks. The *Institute of Electrical & Electronic Engineers* (IEEE) defined the 802.1x standards. The IEEE also defined *802.11*, which is the standard for wireless networks, also called Wireless Fidelity (or *Wi-Fi*). Security standards for wireless networks are defined by 802.11i, a draft standard. A subset of 802.11i is currently used in networking devices as *Wi-Fi Protected Access* (WPA). WPA client applications on host machines enable network users to communicate with APs to

[13] An AP is the point-of-presence for a wireless network.

negotiate security protocols for authentication and authorization. Figure 2.4-1 below shows the authentication and authorization process for a network user. The user communicates to an AP via a WPA client using EAP-over-LAN (called EAPOL). The AP consults a RADIUS server for approval using the RADIUS protocol. The RADIUS server responds to the AP and the AP returns an EAP "success" or "failure" to the WPA client.

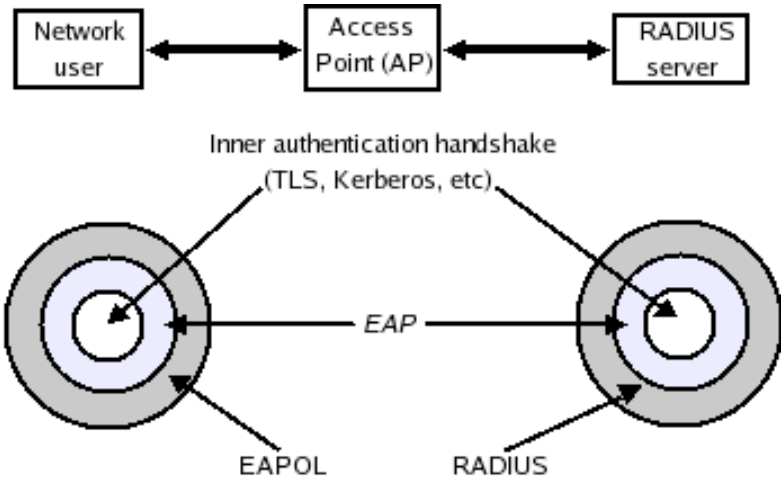

Figure 2.4-1: A network user requests permission to join the network. In 802.1x terminology the network user is a *supplicant*, the AP is an *authenticator*, and the RADIUS server as an *authentication server*. The user and AP communicate by EAPOL while the AP and RADIUS use the RADIUS protocol. Both protocols encapsulate EAP messages that, in turn, encapsulate some other authentication protocol such as TLS. This figure is a re-creation of a diagram from Goffee's thesis on the same subject [Goff04]. Kwang-Hyun Baek put forward the concept for the diagram.

The Greenpass project can be broadly divided into two components: the RADIUS server, and the client tools.

2.4.1 Greenpass RADIUS Server

The Greenpass RADIUS server [Kim04] is built from the open source RADIUS implementation, FreeRadius [FreeR]. Using an open source implementation has allowed the Greenpass team to add extra functionality to the server to handle SPKI certificate chains. Unfortunately EAP-TLS can handle the transmission of X.509 authentication certificates but not SPKI certificates. This restriction means that in their unmodified forms, neither AP, nor client machine, nor RADIUS server can handle SPKI authentication certificates. Rather than require new software, the project presents a solution in the form of a SPKI cache that the RADIUS server can consult for validating a guest. Guests present SPKI certificate chains to the AP, via their browsers (a certificate chain is sent as a cookie), and then present their X.509 Certificates by EAP-TLS. The SPKI cert is added to the certificate cache mentioned earlier. When the Greenpass RADIUS server receives a certificate, it first determines if it recognizes the CA for the certificate. If it does not, it refers to the SPKI certificate cache. If the RADIUS server can find a chain linking the requestor's public key to some trust root (a certificate signed by a trusted CA), then the RADIUS server considers the guest authorized, and informs the AP that forwarded the request. If no valid chain is found for the guest, then the guest is considered unauthorized so the AP redirects the guest to a restricted VLAN, as explained previously.

2.4.2 Greenpass Client Tools

The Greenpass Client Tools [Goff04] consist of all the interfaces that the delegator and guest interact with before a guest is granted access onto the network. These include the Greenpass web application and Python scripts that receive and process a guest's request, as well as the caches that hold the SPKI certificates for authorization. The client tools component is outlined below:

1. Greenpass Web applications: This consists of a set of introductory web pages accessible from the restricted VLAN; the delegation web page that allows delegators to examine requests, compare guests' public keys, and create new SPKI chains for guests' future authentication; and a front page that allows delegators and guests to determine their status in the delegation process. The front page tells guests where to go to get authorization, as well as provides them with the option of seeing their SPKI certificate chains in human readable form or deleting all Greenpass related cookies from their browsers.
2. Enrollment CA: For guests who do not have public keys (do not have X.509 certificates), the introductory page works with a python CGI script to generate a temporary X.509 certificate that the guest's web browser installs in the browser's keystore. This 'dummy' certificate is created to provide the guest with a means of transmitting a public key over EAP-TLS that can be used to authenticate the guest.
3. There are two caches provided as part of the client tools: an introductory cache and an authorization cache. The introductory cache stores guests' X.509 certificates until a delegator views a guest's request via the delegation page. The authorization cache stores SPKI certificates and provides functions to the RADIUS server for verifying *trust paths* (chain of certificates to some trust root) for guests' public keys, verifying

a guest's status (authorized or unauthorized), and adding new certificates. These functions are available by *XML-RPC*, a set or protocols that allows an application on one machine to call functions in another application, perhaps on some other machine.

2.4.3 Greenpass guest access procedure

The steps for providing guest access under the Greenpass approach can be summarized as follows:

1. A guest attempts to join the network and must present an X.509 certificate to an AP. If the guest has a certificate, the AP obtains this certificate via EAP-TLS and forwards it to the Greenpass RADIUS server for authentication. If the RADIUS server recognizes the CA for the certificate or can find a trust path in the authorization cache it returns affirmative to the AP, which then lets the guest onto the network. Otherwise either the RADIUS server cannot find a trust path or the client has no certificate and is shunted off to a restricted VLAN.
2. On the restricted VLAN the client can only access the Greenpass front page (via a web browser) that explains how the guest may obtain authorization, or the introduction page where the guest may enroll for an X.509 certificate if they do not have one. At the front page the guest's browser is asked for an X.509 certificate. The Greenpass web application passes this certificate to the introductory cache for a delegator to review it.
3. A delegator views the guest request via a web browser and approves it. This creates a SPKI chain from the delegator to the guest, and thus provides a trust path from the

guest to the delegator's trust root. The new SPKI chain is then added to the authorization cache.

4. On the front page the guest is notified that her status is now authorized and is presented with the SPKI chain as an HTTP cookie. The guest may then reattempt to join the network. By this time, the guest has an X.509 certificate (it might be a dummy one created from the Greenpass introduction page), and the AP forwards this to the RADIUS server. The RADIUS server will find a new SPKI chain with a valid trust path and so instruct the AP to allow the guest onto the network.
5. The SPKI chain on the guest's machine (as an HTTP cookie) is used to refresh the authorization cache when it expires.

The Greenpass project places a number of requirements on a wireless network user's machine but most of these should not be a problem for many network users:

1. The machine must support EAP-TLS wireless authentication and possess a WPA client. While most operating systems (e.g. Windows XP) provide this application, earlier versions of Mac OS X (< X.3) do not have a built-in WPA 802.1x client.
2. The user's browser must have cookies enabled, and must support client-side SSL. Most standard browsers support enabling and disabling cookies but not all support client-side SSL (Internet Explorer 5.2 on Mac OS X still doesn't for instance). Fortunately, each platform has at least one freeware browser that supports client-side SSL (Mozilla browsers support it).

Chapter 3

The Access Control Service

3.1 Objectives

In constructing the model for the ACS I considered the following ideals:

1. An implementation of the ACS should place few or no software constraints on users' host machines, besides the need for a VPN client that they would use for network access. This was desirable because one of the objectives of the project was to reduce demands on remote network users to find machines that possess one capability or the other.
2. An implementation should provide some means by which administrators could specify policies and privileges. Furthermore, if an administrator was not trying to restrict a resource's users to a particular application, platform, or location then these details about a user should be insignificant to the user's ability to gain access.
3. An implementation should build on current technology (rather than attempt to replace it). This objective means that implementing the ACS should be feasible and affordable for an institution.

3.2 The Model

The access control service (ACS) is made of four primary components: a VPN service, a database service, an accounting and mediating service, and a web service. The function of each service is listed below:

I. VPN Service

- Provides a gateway to the VPN as well as authentication for remote users.

II. Database service

- Stores account information about each VPN user as well as their privileges, as specified by administrators.

III. Accounting and mediating service

- Monitors VPN users' traffic and identifies when users attempt to connect to resources.
- Establishes connections to resources on behalf of valid users.
- Logs users network activities.

IV. Web Service

- Provides an interface through which:

  1. Network administrators may edit users and user groups

  2. Resource administrators may edit user and group privileges for their resources as well as specify how the accounting and mediating service would mediate between the resources' access control systems and the users.

  3. Users may view their account information and privileges.

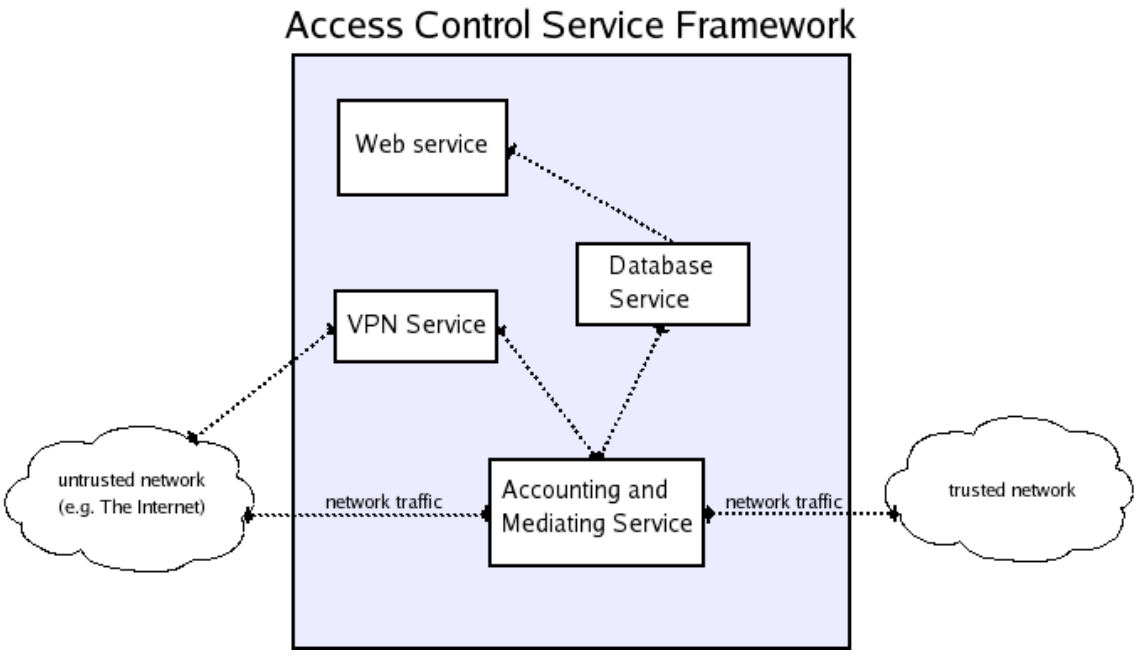

Figure 3.2-1: Various components of the ACS Framework.

The ACS' components interact as follows:

1. The web service provides web pages for users and administrators. The web service stores specifications made by administrators with the database service.
2. The VPN service accepts remote users' network connections and notifies the accounting and mediating service
3. The accounting and mediating service identifies the privileges and specification for each connected user by querying the database service.
4. The accounting and mediating service then sets up rules to track the network user's traffic from the VPN gateway to specific resources on the network. These rules will allow the service to be notified if the user tries to initiate a connection for which the ACS has instructions to mediate.

5. When the user initiates one of these connections, the accounting and mediating service will establish a connection to the resource using the specifications associated with that user. It will then route the users traffic to that resource through the established connection.
6. The accounting and mediating service will then begin logging information about the users activity on that connection (e.g. time of connection and time of disconnection).

3.3 Implementation

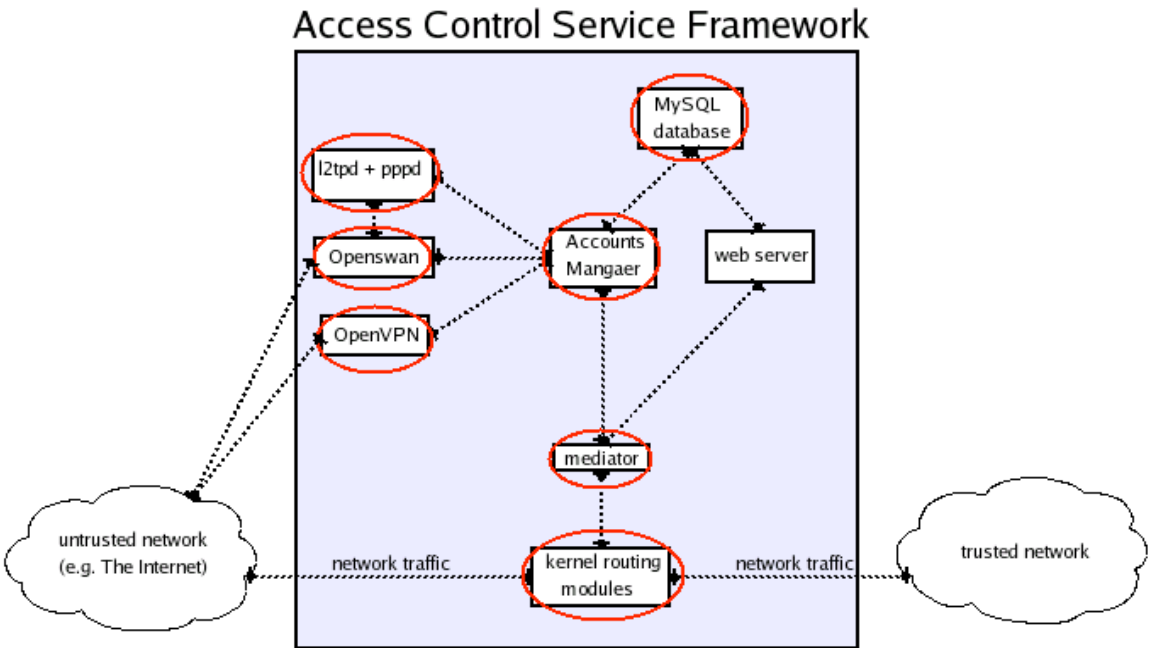

Figure 3.3-1: Design for ACS prototype. The circled components are those installed or programmed in my implementation. The dotted lines between components represent communication, usually by passing messages.

My implementation of the ACS (Figure 3.3-1) is not a fully functional version, but a prototype of what I envision the actual service to be. I built this prototype to aid my proof of concept and to better understand what issues its construction might present. I implemented the access control service framework with a Pentium 4 Dell Inspiron running the Fedora Core 1 distribution of Linux. I chose Linux because of the abundance of open-source solutions for the various components that I thought would be necessary for the service. I built Linux kernel 2.4.22 and downloaded and installed all the other components discussed here. I configured a MySQL database for the database service and installed and configured Openswan [SWANb] and OpenVPN [Yon04] for the VPN service. My implementation of the ACS uses IP address to keep track of a user's activity and to identify resources on the network.

I wrote two programs to provide the accounting and mediating service (see Figure 3.3-1):

1. A Java database manager called the Accounts Manager (AM)
2. A C++ program called the mediator.

In all of the design choices I considered efficiency because I was aware that packet processing at the application level (i.e. outside the kernel) could impact network throughput [And03] [CB97]. It seemed necessary to use identification numbers, rather than IP addresses or hostnames, because remote users might not connect from the same site on each visit to the network. Additionally, unsigned integer comparisons were more efficient than these alternatives. These ID numbers are primary keys in the MySQL database and are used for comparisons and information retrieval within the mediator

program as well. In the MySQL database these ids are mapped to usernames and public key *fingerprints*. Each fingerprint is a message digest of a user's public key. Message digest functions, such as MD5 and SHA1, provide unique hashes for unique messages (in this case, public keys). I used MD5 as the digest function for calculating a user's fingerprint. Once users have connected to the ACS and their id numbers have been established, the ACS builds mappings between id numbers and IP addresses for the duration of the users' connections. These mappings allow the ACS to resolve packet ownership for routing and mediating purposes.

3.3.1 The VPN Service

The VPN service supports IPSec, IPSec/L2TP and SSL VPN connections. My implementation of the ACS uses two different open-source VPN servers to achieve this: Openswan (for IPSec and IPSec/L2TP) and OpenVPN. A kernel module, called KLIPS, provides IPSec support for traditional (IPSec) VPN tunneling. I used a user-land L2TP daemon (l2tpd), and Linux's pppd to add L2TP functionality to this service, if a user needs it. Jacco De Leuuw has documented how to setup an IPSec/L2TP Linux VPN server using open source tools [DeL02]. Figure 3.3.1-1 is a schematic of the interaction between the various pieces that make up an IPSec VPN server.

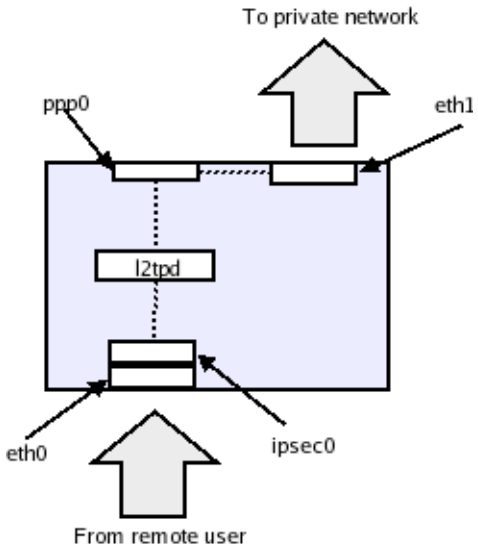

Figure 3.3.1-1: Linux VPN Server setup using pppd, user-land l2tp daemon and Openswan IPSec tool. Reproduced from a similar diagram on De Leuuw's website [DeL02].

There are several open source implementations of IPSec tools for Linux including Freeswan [SWANa], Openswan [SWANb], and Strongswan [SWANc]. I chose Openswan for my implementation though I also experimented with Freeswan and Strongswan. Freeswan is the predecessor of Openswan and Strongswan so it lacks a few features that make configuring the tool as a server a bit easier in its successors; it is somewhat easier to set up Strongswan and Openswan to accept connections without prior knowledge of client configurations. Strongswan and Openswan are very similar in their core functionality but differ in parts of their security focus; Strongswan supports a few more PKI and X.509 certificate functionalities than Openswan. The documentation for setting up a Linux VPN server [DeL02] tabulates the major differences between the IPSec tools. For the purpose of my prototype, Openswan was as equally a good choice as Strongswan.

I built an Openswan 2.2-1 IPSec server for Linux kernel 2.4.22. Linux kernels 2.4.x and earlier require the KLIPS kernel patch in order to handle IPSec (for Kernel 2.6.x this functionality is built in as *26sec*). KLIPS also provides a virtual device interface for processing IP packets. This allows processes on the machine to refer to each IPSec tunnel by a device name much like they would *eth0* or *eth1*. KLIPS uses ipsec0, ipsec1, etc for tunnels. The device driver can handle multiple tunnels under one device name. In my implementation all client tunnels are accessible via ipsec0 (which corresponds to eth0) and all tunnels within the private network are accessible via ipsec1 (which corresponds to eth1). My implementation uses Linux's iptables [NTFLTR] for routing packets from one service to another (e.g. for routing packets between Openswan and l2tpd).

Openswan also provides an IKE daemon called Pluto, which handles the encryption key exchange using a configurable set of algorithms and hash functions (Openswan 2.2 defaults to 3DES-MD5). I configured Openswan to require user certificates for authentication.

I created a certificate authority (CA) for the ACS and generated and signed certificates for all the machines I used in with my prototype. I created the CA and certificates with OpenSSL [OpenSSL]. On my iBook, I also tried out a free IPSec GUI application, Ipsecuritas [LBTMO], which has support for certificates (The CISCO Mac VPN client also supports certificates). Ipsecuritas does only IPSec VPN connections, and the Panther (OS X.3) VPN client does only IPSec/ L2TP VPN connections. I used Ipsecuritas to verify that the VPN service worked for either flavor of tunneling.

Additionally, I set up OpenVPN on the ACS so that the machine could accept SSL VPN connections as well.

A traditional VPN sever has two network interface cards (NICs), one reachable through the Internet (or over the less trusted network) and the other on the more trusted network. Because L2TP/IPSec uses PPP, a client of the VPN server may be assigned a local IP address after authentication (i.e. an address on the more trusted network) by a DHCP server. With purely IPSec VPNs, the client can optionally be masqueraded[14]. Iptables provides masquerading as part of its NAT package. Iptables' NAT has fixes for most of the common problems associated with NAT so protocols like FTP, IRC, and Real Audio, can work with hosts doing NAT via iptables [NTFLTR].

Linux (and many other operating systems) allow users to enable IP forwarding[15] between subnets. This functionality is at the core of most Linux open source VPN implementations because it allows packets coming to the server from a less trusted network (or the Internet) to be able to reach the more trusted one solely at the discretion of the server. My ACS prototype has two NICs, each on a different Dartmouth subnet. In my implementation neither subnet was more trusted than the other, and both were reachable from the Internet. However, the same setup applies to the traditional set up (i.e. where one NIC is on a more trusted network than the other).

3.3.2 The Accounts Manager

[14] Masquerading is a form of network address translation (NAT). Please see Chapter 2 for an explanation of masquerading and NAT.
[15] IP forwarding is the same thing as 'bridging' in Windows (for readers familiar with the 'bridged connections' option in Window's network configuration).

I wrote a Java application to interface with the MySQL database. The accounts manager (AM) is responsible for mapping a newly connected user's information to a pre-assigned id number and informing the mediator. For example, when Pluto concludes the IKE phase of an IPSec authentication with a user it should forward the user's IP address and certificate to the accounts manager[16]. The AM retrieves the user's public key from the certificate and calculates the MD5 fingerprint of the key. It then queries the database for an id-to-fingerprint mapping with the given fingerprint. If it can find such a mapping then the user has an associated object in the mediator's database. Once the AM has resolved the user's id it sends a message telling the mediator that the user is now connected with the given IP address. The AM sends the message using the ACS syntax explained in section 3.3.4. An example is given below:

```
CLIENT 381 {
     HOST 129.170.210.183
}
CLIENT 381 UP
```

Example 3.3.2-1: Sample notification message from AM to mediator service.

The message tells the mediator to update its database for a client with an id 381, and assign that client an IP address of 129.170.210.183. The last line 'activates' the client object. Upon receiving this message the mediator can add routing rules for the particular client and perform a network address translation on the client's traffic if needed. Activating a client object is an indication that the mediator should monitor the ACS'

[16] Pluto does not naturally provide this functionality; it requires a patch to do this.

network traffic for data from or to the corresponding network user, based on the privileges of the user. Activating a resource object, on the other hand, indicates that the mediator should consider the resource available for access on the network.

Under my current design, if the user connects using IPSec/L2TP then after the IPSec server authenticates the user the l2TP server would forward the username and IP address assigned to the user. The AM is able to look up the user's id by username. If it can find such a mapping, it would construct a message for the mediator as described above. The same applies to OpenVPN.

3.3.3 The Mediator

The mediator manages the state of each user's connections to various network resources. In my prototype, the mediator loads up policies for each user from a file at start time, and creates objects for users and resources to represent their current states at any given time. During regular execution, the mediator is usually waiting for an event to occur. It waits on two classes of events: (1) messages from either the AM for updates to its objects. (2) Messages about clients' attempts to connect to network resources. The mediator has a dedicated port for receiving updates to its internal objects structure. The mediator uses the conntrack module, provided by iptables, to identify a connection request from a client (see Chapter 2 for an explanation of the iptables state tracking module).

I designed the mediator with loadable policy-handling modules in mind. I felt this approach would provide extensibility to the mediator's current policy-handling

capability. My implementation of the ACS handles IP-filtering policies and can proxy other VPN connections to other subnets or hosts. When a policy-handler is invoked for the first time, it applies iptables rules to queue client packets that are destined for target secure servers. Iptables provides hooks for queuing packets for userland applications (the mediator in this case) through the Netfilter library, libipq, and iptables' QUEUE target (See section 2.3.2: Iptables). I make all adjustments to the iptables rule-set using shell scripts invoked by the mediator through system calls. These shell scripts receive arguments for a particular client and modify one or more chains and tables with these arguments.

```
iptables –t nat –A POSTROUTING –s $CLIENT –d $SERVER –j SNAT –to
$NAT_ADDRESS
```

Example 3.3.3-1: Sample iptables rule for 'source' natting. The $CLIENT parameter is the IP address of the network user, the $SERVER parameter is the IP address of the target resource on the VPN, and $NAT_ADDRESS is the IP address to which the ACS will map the user's traffic for the target resource.

Once the mediator receives a user's connection-request packet it identifies the user and destination resource and checks its database to determine if a policy handler is available for the user's connection. If no policy handler exists then there is no authorized connection for the user to the resource and the ACS simply removes the queuing rule so that packets of this nature are left to the target resource's discretion (i.e. the resource's access control policies). If, on the other hand, such a handler exists, then the mediator calls the handler with arguments stored with the handler's declaration, and instructs it to execute because a connection attempt has been made. The arguments passed to the policy

handler differ based on the particular handler and allow any policy handler to discriminate between clients by specifying which credentials the ACS should use for authentication. In all cases, the ACS has a pool of credentials (certificates and matching private keys, IP addresses, etc), in its name. The policy handler uses one of these credentials to authenticate with the target server. When the mediator program ceases execution, or the C++ client objects are destroyed, the handler is called again to do the necessary clean up (including removing Iptables rules). I will explain the operation of the policy handler for two policies that the prototype currently handles:

I. IP-Filtering Handler

In this policy, the target server allows connections from hosts with a specific type of IP address (or IP addresses from a specific range/pool). The server allows the ACS to use a subset of its accepted IP pool for remote clients. The handler for this policy keeps a table of these IP addresses and records those that are in use by a client at any given time. The argument passed to the handler is a reference to the current table entries (as well as a C-struct containing information on the user and server that is passed to all policy handlers). When the handler is informed that a connection request has been made it finds the next available IP address and adds an iptables rule so that all subsequent traffic from the user to the server will be 'natted' with the selected IP address. If there are no more IP addresses then the ACS has ran out of available connections to the target server and the

handler will issue an *NF_DROP*[17] directive for the packet. The ACS will, therefore, drop all packets to that server from users without an established connection until other users free up their connections. This behavior is configurable: alternatively the handler can be instructed to reuse IP addresses when entries in the table ran out. The latter option allows the ACS to create an unlimited number of connections to a resource. The former configuration will limit the number of connections from the ACS and so can prevent possible *denial-of-service*[18] attacks through the service.

II. VPN Handler

In this policy, the target server is a VPN concentrator/server to some subnet, perhaps within the same network as the ACS. The server machine could even be another ACS. The ACS has a certificate that is signed by a trusted CA of this target server. The mediator and uses this certificate to setup a VPN tunnel (IPSec or SSL) to the target server. Before the tunnel is set up, the ACS blocks all forwarded packets from going through the tunnel (without this initialization, other ACS clients may access the target server and the restricted subnet behind it). When the handler is invoked (upon a connection request by the client), the handler adds a routing rule to the route chain to exempt the client's forwarded traffic from the previous restriction rule. Subsequently, these packets are automatically routed through the VPN tunnel, between the ACS and target server, to the secure subnet behind the server.

[17] NF_DROP and NF_ACCEPT are flags that a process sends to the iptables queuing handler to decided the fate of a queued packet (i.e. to drop or accept the packet).
[18] Denial-of-service (DOS) occurs when a server can no longer accept connections from users because it is already maintaining the maximum number of connections it can support. By restricting the ACS to a number of connections at a time, a server will always have connections available to users who are not requesting access through the ACS.

The ACS maintains state on all users and resources through client and server objects. Server and client objects can be 'activated' or 'deactivated.' A client object corresponds to a particular network user. Activating such an object means that the user is currently connected to the ACS. When the user disconnects, the AM informs the mediator and the corresponding client object is deactivated. Server objects represent resources on the more trusted (private) network. For these objects, activation means that the resource is available to receive access requests from network users. It also means that the administrator of the corresponding resource wants the ACS to mediate on behalf of the resource's authorized users. If a server object is 'deactivated' then the administrator does not want the ACS to do any mediation for users accessing that resource. This might also indicate that the resource is temporarily unavailable (or offline).

3.3.4 Specifying policies

I designed a simple grammar for describing resources, users, the privileges the users have, the policies used for authenticating the user, and which credentials the ACS would employ to proxy the user. In the absence of a web service in my implementation, the grammar serves as a means for specifying policies. The syntax uses a few keywords and delimiters to mark the start and end of the definition of an object. A parser for the grammar recognizes three keywords for specifying objects: *server*, *client*, *and conn*. Each of these implies the declaration of an object if the object does not already exist in the database, or the modification of the object if the object has previously been created. The

ACS parser recognizes the delimiters '{' and '}' as markers for the start and end of the attributes of an object, and the end of line character as a meta-symbol before a new declaration. Example 3.3.4-1 presents an example of instructions using the ACS syntax.

```
SERVER server-id {
      HOST IP-address
      CLIENT client-id {
            CONN {
                  TYPE IP-FILTER {
                        SUBNET first-IP-address last-IP-address
                        UNIQUE
                  }
            }
      }
}
```

Example 3.3.4-1: Sample Server Declaration for ACS input file. All italicized words are variables; an actually policy file (or message) would have numeric constants in their place. The variables are included here to clarify the meaning of these constants. All terms in bold face are keywords that the ACS parser recognizes. *Subnet* and *unique* are underlined because a 'special' parser is required to recognize these keywords. These 'special' parsers are explained later in this chapter.

The specification in Example 3.3.4-1 declares a server with a particular identification number and IP address. It then declares a client, also with a particular ID number, and specifies that the client will be authenticated by an IP-based filtering policy and so should be assigned an IP address from a range between *first-IP-address* and *last-IP-address*. Like the policy-handling mechanism of the mediator, the mediator's parser

was also designed with modules in mind. The parser itself must invoke special, more specific parsers for various policies. Thus when the parser encounters the *type* keyword. It looks up the appropriate special parser (in this case the IP-Filter parser), which parses further specification between the '{' and '}' delimiters. Again this modularity is intended to provide extensibility to the ACS, allowing it to parse other policies that may be added to its capabilities in the future.

In Example 3.3.4-1, the IP-Filter parser recognizes the *subnet* keyword and stores the upper and lower bounds that define the address pool. The ACS has a table with entries for all the IP addresses it can use for its clients and the IP -Filter handler will search this table within the specified bounds when it needs to assign a user an IP address. The *unique* keyword tells the policy handler not to use an IP address that is already in use. This might be desirable if an administrator wishes to restrict the number of connections the ACS can make on behalf of its users. Without this keyword, the handler will first try to assign a client an available IP address and then pick any non-unique one if all are in use. If a handler picks an IP address for a client object that uses 'unique' keyword then a flag is set against the IP address in the ACS' global table so that other handlers do not reuse the address until it is freed.

3.3.4.1 The ACS Grammar

The parser uses a grammar of the form given in Table 3.3.4-1 below. A more accurate grammar is given in **Appendix A**. I give this simplified version because it captures the essence of ACS grammar and is much easier to explain.

<Input File>

<Declarations>

<Declarations>

<Server Declaration> <Declarations>

<Client Declaration> <Declarations>

NIL

<Server Declaration>

SERVER *uint*

SERVER *uint* **{**<Server Definition>**}**

<Client Declaration>

CLIENT *uint*

CLIENT *uint* **{**<Client Definition>**}**

<Server Definition>

<Client Declaration> <Server Definition>

<Host Declaration> <Server Definition>

NIL

<Client Definition>

<Conn Declaration> <Client Definition>

<Host Declaration> <Client Definition>

NIL

<Host Declaration>

HOST *string*

<Conn Declaration>

CONN {<Conn Definition>**}**

<Conn Definition>

TYPE *uint* **{**<Type Definition>**}**

<Host Declaration> <Conn Definition>

Table 3.3.4-1: Simplified ACS Grammar for policy specifications. The actual ACS grammar is defined for files and messages (passed between processes).

Each rule of the grammar consists of a variable (left justified) followed by several variables and/or terminals that can be substituted for it[19]. Phrases between '<>' represent variables of the grammar, and words in bold face represent terminals. Italicized words represent an entire class of terminals. Thus *uint* is the class of integers greater than or equal to 0 (and no greater than 2^{32}). NIL means no input text. Each variable begins on a

[19] I first came across formatting grammar in this particular way after taking a class with Prof. Bill McKeeman at Dartmouth College.

new line. Thus if '<A> <B>' is the right-hand of a rule, then it represent instructions that take up at least two lines in a valid input file (or message).

The mediator accepts an input file if it can be parsed with the grammar given in Table 3.3.4-1. Parsing is done 'top-down' in the order the file is read. A valid input file contains 0 or more declarations at the *top-level* (i.e. outside of any curly braces) of the file. Each declaration is either for a server or client. Consequently, at the top level the grammar only permits the creation and modification of client and server objects (the complete grammar also allows the deactivation and activation of these objects). Both client and server declarations must begin with the appropriate keyword, and be followed by an identification number. Optionally, these declarations may include a definition for the object that must be between two curly braces. The definitions of both types of objects (servers and clients) may include a host declaration, specifying the IP address of the object.

The definition for a server may include the declaration of one or more client objects. Similarly, the definition for a client may include the declaration of one or more *conn* objects, which are specifications for the authentication policy the client must satisfy when connecting to a given server. A conn declaration must include the definition of the conn object between curly braces. A conn definition may also include an IP address assignment. The IP address within a conn definition specifies the server; the client of the conn object is already known from the scope in which the declaration occurs (conn objects are defined within client object definitions). If the enclosing client definition is within a server definition then the host declaration is optional; the server for the conn object defaults to the server in which the client is being defined. This exclusion is not

specified in the grammar in Table 3.3.3-1 but the parser will reject the input file at runtime if it cannot resolve a server for any given connection. **Appendix A** presents a more complete grammar that my implementation uses.

A conn definition must include a type definition, which is the set of arguments and details for the particular policy. There are no rules for the variable <type definition> in the ACS grammar because the parser does not resolve this variable. As previously explained, the parser invokes a special parser for each specific policy type. The parser maintains a database of parsers that it may invoke to resolve each specific policy. Thus when the main parser encounters the 'type' keyword, it reads the subsequent unsigned integer token as an index into its database of policy parsers. The specialized policy parser will parse the type definition and return control to the main parser. The specialized parser also returns an indication of success, if it is able to obtain a valid type definition, and failure otherwise. Example 3.3.4-1 demonstrates the type definition for an IP-filtering policy.

3.3.4.2 The Web Service (not implemented)

In a full-scale implementation, the web service would provide several web pages for both administrators and network users. Administrators' web pages would allow them to select their resources via web-form objects (e.g. drop down boxes). Administrators would be able to add new resources or modify the policies for those already displayed by the website. They would also be able exclude resources from the ACS' mediation. The web pages would identify each policy by type (e.g. IP-Filter or VPN) and for each type would allow the administrator to edit configurations. For example, for configuring a

VPN's access policy, the website would present an admin with a form to upload a certificate and private key file that the ACS may use to establish a tunnel, and select what type of protocol the resource required for VPN access – IPSec, IPSec/L2TP, or SSL. For each resource, the Administrator would also be able add or drop which network users she has authorized to access the resource, and perhaps what type of access privileges these users may have: For example a user might have administrative access or restricted access, etc. In this case the administrator would have to provide the ACS with a different set of credentials for each access privilege type.

I created the ACS syntax for specifying policies in the absence of a web service. However, I could have added the web service to my implementation by providing scripts to translate specifications made on a web form into ACS syntax; once the web service generates the messages, it would send these messages to the mediator the same way the AM currently does.

Web Form Specification	ACS Syntax equivalent
Prevent the ACS from mediating between users and a particular resource	SERVER 32 DOWN

Add a server as a network resource.
Specify the name and IP address of the server. State that the server uses SSL VPN access control.
Upload a private key file. Upload a certificate file (The ACS will present these
	SERVER 35 {
	HOST 129.170.214.90
	TYPE VPN_SSL {
	CERT /usr/share/acs/certs/cert.pem
	KEY /usr/share/acs/secrets/key.pem
	}

to gain access for a privileged user) State that the resource is active (i.e. its available to receive connections for network users)	} SERVER 35 UP

Authorize the ACS to mediate on behalf of a client to obtain access to the server defined above. The client will use the default settings for establishing a tunnel to the server.	CLIENT 62 {
	CONN {
	HOST 129.170.214.90
	TYPE VPN_SSL {
	}
	}
	}

Table 3.3.4.2: The web service could generate ACS syntax messages from information gathered from the web forms. The entries above are examples of this translation.

The web service would also provide web pages for network users to view their privileges and any other relevant account information.

3.4 Putting it Together

My prototype does not include the web service component identified in the model for this framework. Instead, specifications are made by means of a syntax that I designed for allowing the various components to communicate. My prototype also implements the accounting and mediating service as two separate programs: the Accounts Manager and the mediator.

The prototype is designed as follows:

1. The VPN services comprise of OpenVPN, Openswan, l2tpd, and ppd. When a user connects to the VPN service, it will give the AM some identifying information about the client (username or X.509 certificate) and current IP address.
2. The AM looks up the id number of the user with the given identification information (username or public key fingerprint) by querying the MySQL database.
3. The AM constructs a message, using ACS syntax, telling the mediator that the corresponding client object should be activated, as well as providing the user's current IP address.
4. With the given IP address, the mediator can add routing rules to queue certain packets from the user's traffic based on the destination of these packets (which resources the client is trying to access). These packets represent connection attempts.
5. When the mediator detects a connection attempt (i.e. it receives a queued packet) it establishes a connection with the resource and allows the user to access the resource through this established connection. The mediator establishes connections on behalf of a user based on the resource administrator's specifications.
6. When a network user disconnects from the ACS, the VPN components will send another message to the AM with some identifying information for the user (in this case an IP address). The AM looks up the user's id with the given information and constructs a message telling the mediator to deactivate the corresponding client

object. The VPN services disconnect idle connections automatically so inactive users do not remain connected to the system.

3.5 Summary

The ACS model presented in this chapter and my prototype implementation have demonstrated the following:

1. The ACS is able to reduce unintended access denial of authorized users by mediating connections between users and resources.
2. The ACS endeavors to maintain the security standards of the various network resources[20] because it uses VPN technology, which provides authentication, confidentiality, and data integrity; users that benefit from the ACS are not only authenticated but are also authorized to access the resources that the ACS enables them to.
3. Because the ACS forms an intermediary between resources and network users, it provides a centralized system for administrators to control access to their resources in a way that captures the administrator's intentions for securing the resource. This point also suggests that resources network-wide may benefit from new technology introduced to the ACS framework. Chapter 4 discusses how ACS could be upgraded to work with Greenpass tools and so enable resources on the VPN to support decentralized delegated access without further configuration.

[20] In the sense that it provides direct access to only authorized users. This does not guarantee the integrity of the data after the host receives it; the ACS cannot vouch for the security of the host the user is using (see Chapter 4 for further discussions)

3.6 Concluding Remarks

My prototype deviates somewhat from the model presented at the beginning of this chapter. This discrepancy is due to the fact that I was able to refine the model only after considering some of the decisions and their effects on the prototype. For this reason, the prototype could have had a single program for the accounting and mediating service. With a single program, the service could read specifications directly from the database service and there would be no need for a policy specification grammar; while the web service and the accounting and mediating service would have agree on a format for storing and retrieving specifications, this format would be a lot simpler than the ACS syntax that my prototype currently uses.

Besides establishing a proof of concept, I embarked on implementing a prototype as a way to better understanding what putting together a framework like the ACS would entail. I believe that my prototype has accomplished just that.

Chapter 4

Discussions

This chapter answers some of the questions that arise from my implementation. I will focus on the issues of usability, efficiency, and security, as well as discuss the improvements to the model, which are necessary for addressing these issues. Before this, I will explain how the ACS may be made to work with Greenpass tools for introducing delegated access on the network.

4.1 Delegation

The ACS framework allows multiple resources to be managed centrally. In particular, enabling the ACS framework to understand delegated access will enable all services on the network to benefit from delegated access. The ACS model can be upgraded to support Greenpass tools including RADIUS authentication to the VPN service and implementing the accounting and mediating service to work with the authentication cache. These suggestions can also be directly applied to my prototype and sections 4.1.1 and 4.1.2 explain what modifications need to be made to enable decentralized delegated access for network resources.

4.1.1 Adding RADIUS-server support

1. Openswan supports *Xauth*. Xauth is a program used to edit and display configuration information for connecting to an *X server* [Ful05]. Xauth can be made to use *PAM* (Pluggable Authentication Modules), an authentication system that controls access to Red Hat Linux [RHLb]. PAM can also be configured to use RADIUS authentication so that client certificates can be forwarded to a RADIUS server for validation. It should be noted that Openswan developers do not consider the use of PAM with Xauth very safe [SWANd]; only recently, iDefense published a buffer-overflow vulnerability in Openswan when compiled for Xauth/PAM authentication [iDFNS05]. In light of this I hope to find an alternative for authenticating IPSec VPN connections.

2. Linux PPP can be configured to do RADIUS authentication. This functionality is built-in for pppd 2.4.2 and later versions [DeL02]. For earlier versions of pppd there are plug-ins available. Because the L2TP/IPSec VPN setup uses Linux's pppd, configuring pppd to use a RADIUS server will provide RADIUS authentication for the VPN connection.

OpenVPN also has a PAM plug-in that can be configured to use RADIUS. This is similar to approach (1) above. Alternatively, the OpenVPN configuration file provides a number of instructions that allow scripts to handle user authentication [Yon04]. These scripts can be used to send a certificate to a RADIUS server for authentication.

4.1.2 Adding delegated-access support

In order to extend the prototype's service to delegated users, the AM and mediator must be modified to understand delegated access: If the AM receives a certificate and cannot find an id assignment for the user, for instance, then it should query the Greenpass RADIUS server's authentication cache [Goff04] (see section 2.4 for information about the authentication cache). The Greenpass authentication cache API can be modified to return entire SPKI chains for a given user's public key. Greenpass currently does not support delegating a subset of a user's privileges[21], however, there have been plans to incorporate such a scheme and it is likely that the tag extensions of a guest's SPKI chain will convey information about delegated privileges

The prototype's grammar can be extended to use three keywords, *guest*, *until* and *like* for representing transient users and privileges. Table 4.1.1-1 shows how these keywords would work.

Syntax	Interpretation

GUEST 5 {
LIKE CLIENT 82
UNTIL 05/01/2005 18:00:00
HOST 190.181.183.14
}
	Create a client object that should be treated as *transient*; it will 'expire' at the end of the date given by the 'until' instruction. Assign this client the IP address 190.181.183.14 and accord it all the privileges of the client with id 82 until it expires.

GUEST 6 {	Create a transient client object that is valid

[21] It has not been necessary to do this yet since until now the only privilege in question has been network access and a user either has it or does not

HOST 190.181.183.15
UNTIL 05/01/2005 18:00:00
CONN {
HOST 129.170.253.74
LIKE CLIENT 82
}
}
	until 6pm on May 1st 2005 and with IP address 190.181.183.15. Use specifications for client object 82 only for connections to a server object with IP address 129.170.253.74.

Table 4.1.1-1: Interpretation of syntax using GUEST, UNTIL, and LIKE keywords.

The 'guest' keyword is much like the 'client' keyword introduced in Chapter 3. The difference is that the client object created by a 'guest' instruction persists only until the date and time specified by the 'until' instruction. After this time, the corresponding user's privileges expire. The client object is, therefore, *transient*.

The 'like' keyword instructs the mediator to use the specifications of some other client object for the current client object. The other object corresponds to the delegator for the user whose object uses the 'like' keyword.

The recommendations presented in this section would allow ACS to work with Greenpass tools for delegated access. With this enhancement to the model, users will be able to delegate their access authorization if they have delegation privileges. The end result would be a system whereby users can grant temporary access to network resources in a decentralized fashion.

4.2. Usability and Efficiency

4.2.1 Handling larger client load

The ACS does not discriminate based on a user's location, i.e. it can accept user connections from within the VPN. This could allow local (internal) network users to benefit from the ACS' services as well. Figure 4.2.1-1 shows a network topology with internal (local) and remote users benefiting from the ACS's services.

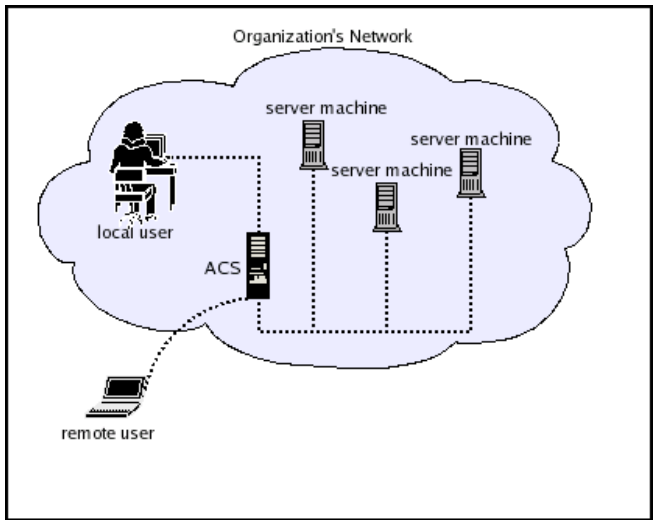

Figure 4.2.1-1: ACS supporting remote and local network users.

At the same time the ACS might potentially have to serve as many people as there are network users. A network may, therefore, need to support multiple clones of the various components; an overloaded accounting and mediating service, for instance, could use port forwarding to reroute all connections to a less busy clone and so balance user load.

Large user populations may also create an inconvenience if administrators have to add and remove users one at a time for each of their resources. Therefore, another logical improvement to the current implementation would be to include support for user and resource groups.

4.2.2 Increasing the protocol working set

My prototype of the ACS has been concerned with IPSec, SSL, IPSec/L2TP VPN policies and IP-based firewall policies. These policies all control access at the network layer (or lower). However, the ACS model does not restrict service to only network layer access. Consequently, above the network layer (specifically at the application layer), it might be necessary to proxy access on an application-by-application basis in order to extend the implementations working set of policies. I currently have plans to provide a web proxy approach to handling client-side PKI policies for web servers on the network. I hope to have enough information about a solution, and a possible implementation, shortly.

4.3 Security Considerations for the ACS Framework

In my research, I identified that the ACS must be secured against two primary classes of adversaries: unauthorized network users, and malicious authorized users. The former have no permission to access the VPN. The latter have authorization to connect to

the VPN but may misuse their privileges on the network. Alternatively, a malicious network user could *hijack* other users' authorizations and misuse their network privileges. I also identified that the ACS stores two classes of sensitive information (1) Account information for administrators and users (2) Private keys and shared secrets that belong to the ACS. In the ACS model this sensitive information is stored with database service. This makes the database service an integral part of the framework's *trusted computing base* (TCB) – i.e. the security policies that govern the database service determine the reliability of sensitive information within framework.

4.3.1 Securing access to the ACS

First, a number of policies could be used for securing access from the web service and VPN service. For instance, the web interface for administrators could require client-side SSL authentication as well as usernames and password because client-side SSL authentication uses PKI, which is considered more secure than server-only SSL authentication or username-password schemes [Hos04].

Second, all network ports accessible from the less trusted network (except those for the web server and VPN server) could be firewalled. This would leave the VPN and web services as the only entryway for access from the less trusted network. Because both services authenticate connections that they receive, all access from the less trusted network to the ACS or the resources on the network would be authenticated. Consequently, any malicious user who attacks from the less trusted network must have

authorization, either hijacked from another user or provided by a network administrator[22]. With 'best practices'[23], the probability of a malicious user hijacking another's authorization (by stealing their credentials or sessions) could be minimized:

- The ACS could rely more on PKI (private key and certificate) authentication than passwords. Where passwords are used, certificates could also be required for additional security. For example, for authenticating administrators on the web service.
- Users should not leave open sessions idle on public machines when they are done accessing resources. The ACS could help out by causing idle sessions to time-out (OpenVPN and Openswan already provide this functionality).
- 'Strong' passwords could be used for accessing accounts on the web server; the ACS could ensure that when users and administrators create passwords, the chosen passwords would not be easily susceptible to dictionary attacks.

Attacks may also be initiated from within the more trusted network or by taking control of processes within the ACS framework. Various policies could be employed to limit such attacks as well. For instance, in order to reduce the damage that an attacker could exact by taking control of a process, the various components of my implementation ran at user privileges on the Linux box. For example, the MySQL database runs under user 'mysql' privileges, while the web server runs as user 'www'. This means that even if both are compromised, they cannot be used to access a substantial amount of system

[22] In practice it is virtually impossible to claim that a system can only be compromised by some means and not by others because the tools and capabilities of malicious network users are constantly improving.
[23] 'Best practices' refers to security measures that are considered most effective in reducing the risk of attacks. Even the most effective measures are not foolproof.

resources. Additionally, on the web interface, the ACS would reduce direct interaction with the database service by providing HTML form-objects to users. It would be the results from these objects that would be used for database queries rather than users' inputs. All processes that send requests to the database service could also *escape* (i.e. quote) their queries first. In my implementation, I followed a few other security measures including removing the 'test' database from MySQL (this is a default database that is usually unprotected), and configuring the server to allow 'root' access from only within the host (i.e. disallowing remote 'root' access). There are several other best practices and security policies for securing a database server [ISS] and any implementation of the ACS should consider employing these.

4.3.2 Auditing network activity

The ACS framework must provide a way for administrators to audit resource access activity. The ACS can trivially log each user's activity because it always maintains state on the connection of each user. Administrators could be provided with these logs, upon request, from their web accounts. Additionally other monitoring tools can be used to filter the logs for inconsistency and suspicious behavior so that administrators might be notified.

4.3.3 Concerns about binding IP addresses to privileges

My prototype of the ACS authenticates a user, determines the user's IP address and creates rules for routing the user's traffic based on this IP address. Similarly, the

mediator looks up privileges based on the IP address of an access request packet that it receives. There are a number of problems with such a design:

- In the introduction to this paper (chapter 1), I mentioned that there were packet fragment attacks and IP spoofing schemes to thwart systems that make decisions based on IP addresses. The current design suggests that malicious users could employ one of above-mentioned techniques to gain access to resources for which they did not have authorization.
- It is not clear how the ACS would differentiate between users connecting from behind a NAT box under the current design; to the mediator, such users would appear to have the same IP addresses.

These potential problems arise from the fact that IP addresses are not uniquely identifying, as are public keys or usernames. Therefore, a mapping from public keys or usernames to IP addresses loses some information about a user. If the ACS is to continue using IP addresses for identifying users then some other information must be available for differentiating between users with the same IP address. I have identified a number of feasible solutions:

Approach 1:

Users connecting from behind the same NAT box usually have different source ports associated with them. NAT boxes typically use these ports to uniquely identify each host (see section 2.2). The ACS could associate users with both IP addresses and port numbers as unique identification.

This approach does not prevent a malicious user from spoofing another user's port number along with the IP address. However, the ACS' VPN sever cannot correctly maintain two tunnels with identical remote IP addresses and port numbers because it would not know which to use for sending reply packets. It is, therefore, arguable that a malicious user cannot successfully spoof both another user's IP address and port number so this approach may be adequate for uniquely identifying network users.

Approach 2:

Netfilter provides a connection-tracking module that can differentiate multiple connections between the same endpoints. The module identifies individual connections based on IP addresses, protocols, ports and other unique identifying information. The module can be modified to include a unique identification number for each related connection (i.e. all connections from the same unique user). The ACS always knows when two users have conflicting IP addresses because two active client objects will map to the same IP address in its database. In such a situation, the ACS could resolve conflicts by consulting the connection-tracking module's logs. The ACS could then determine which user made the request from the unique identification number in the log.

Approach 3:

Netfilter can mark packets at almost every stage of a packet's journey through the kernel's routing system (see section 2.3). The ACS' VPN servers are able to uniquely identify each user even if two users share the same IP address. VPN servers are able to do this because each VPN protocol must necessarily use some unique identification, other

than IP-header information, for each user's traffic. For example, the IPSec protocol identifies each client's packet based on an SPI number, which is resolved before any other header information is used. Similarly, the SSL protocol uses session ids to uniquely identify connections. Thus, of the various components that make up the ACS, the VPN servers can best differentiate between packets coming off the VPN tunnel. In light of this, the servers can be patched to call Netfilter's mark module to mark each packet based on the identified user. The servers will communicate this mark as part of the identification information sent to the AM. This way, the mediator can use both the IP address and the identification mark of each packet to uniquely identify a user.

4.4 Summary

The implementation employed in this research raises a number of security, efficiency and usability concerns. In this chapter, I have presented answers to most of the identified concerns, with a call for further research into any that might not have been addressed here. In many cases, these concerns are specific to the design decisions of my current implementation and not the project's concept (the model) itself. This suggests a need for more rigorous analysis of each design decision and, perhaps, further research into alternative designs for the ACS.

Chapter 5

Related Work

5.1 Napoleon Tools

Napoleon [TOB98] is a policy specification tool for application developers. Napoleon is built on the Role Based Access Control (RBAC) framework, and is primarily for *CORBA* applications. RBAC is designed to aid administrators and developers in specifying network-wide access control policies. Napoleon provides a set of tools that integrate the RBAC functionality. While RBAC provides a way for defining and controlling resource access for network users, it does not work well with legacy applications that were not built with the RBAC model [TOB98]. Such applications require RBAC object representation under Napoleon, and this may be sometimes infeasible. For example, it is not trivial to create an object that captures the complete functionality of an FTP server [TOB98].

5.2 Integrated Secure Communications System (ISCS) Project

The ISCS [Open+05] is an on-going project that aims to provide easy network security management. The ISCS will manage firewalls, VPN tunnels, intrusion detection, virus scanning, application proxying, and user authentication. It currently manages some

but not all of these security features. The ISCS is not a security tool itself; instead it is a management system for other open source security tools. Currently, the underlying tools used are Openswan, Freeswan, Strongswan, iptables, OpenCA, Linux Advanced Routing, OpenSSH, the ISC DHCP server, and the Strongsec DHCP relay. Because of its integrated environment and centralized control to network security, measurements show that the ISCS can cause as much as a 90% reduction in the time taken to configure a network-wide security system.

The ISCS provides a GUI[24] Security Policy Manager (SPM). The GUI allows administrators to write high-level security policies that are translated into low-level rules used in firewalls, at VPN gateways, etc. The SPM maintains a database of *Accessors* (users) and Accessor Groups on one hand, and Resources and Resource Groups on the other. High-level administrative policies define how Accessor and Accessor Groups may access Resources and Resource Groups [Sull04]. The project is fairly new; pre-alpha releases for ISCS have been available since August 2004 and one white paper is available on line [Open+05]. No full release is available at the time of writing this paper.

The ISCS promises to provide some of the same services as the ACS:

(1) Both provide a centralized solution for resource administrators to specify policies for multiple resources.

(2) Both identify resources on one hand, and users on the other, and allow many-to-many mappings between these sets, where the mappings are access control specifications.

(3) Both can provide more modern and technologically superior security to legacy systems that may have little or no security at all.

[24] GUI stands for Graphical User Interface.

The fundamental differences lie in the focuses of the two projects: The ACS aims to make access control management easier for administrators, with an emphasis on facilitating accessibility for users (even where advanced security is employed). The ISCS aims at making it easier for administrators to configure policies for large numbers of security tools, with a focus on allowing corporations to integrate their networks in the event of mergers, partnerships, and acquisitions. The ISCS project appears to target businesses more than any other institutional field, identifying that network security systems of growing businesses undergo large modifications and frequent changes. Additionally, the ACS allows decentralized network access management: Different administrators can specify policies on it that affect only their resources. The ISCS does not provide this kind of decentralized management. Because of this and the differing focus of the project's objectives, the ISCS cannot directly provide solutions to the issues of network accessibility that the ACS seeks to address.

5.3 Globus Project

The Globus Alliance is a consortium of organizations and individuals developing fundamental technologies for the "Grid" [Glob05]. The Grid is a distributed computing infrastructure that enables coordinated resource sharing and problem solving among users across corporate, institutional, and geographic boundaries without sacrificing local autonomy [FKT01] [Fos02]. The Globus Alliance makes all its technology available in the form of the Globus Toolkit. The Toolkit provides a framework for building Grid

applications by providing basic mechanism of communication, security, and data access [FKT97]. It currently contains C and Java libraries, as well as Java and Python web services, for resource monitoring, discovery, and management. The toolkit also provides tools for security and file management across networks.

The Globus project introduced X.509 Proxy Certificates [WFK+04] as part of its security infrastructure. Proxy Certificates are similar, in structure, to regular X.509 certificates; the main difference is that an end user of a regular X.509 can be the issuer of a proxy certificate. This allows users to delegate privileges to other users, machines and services.

The Globus toolkit and Proxy Certificates provide a means for privilege delegation but do not work with legacy network systems that cannot process Proxy certificates. In the absence of other infrastructure, therefore, the toolkit cannot provide solutions to the issues of network access control that this research identifies.

5.4 Now User Filtering Works (NuFW) Project

NuFW is an authentication firewall software suite for Linux [NuFW]. The suite is built on top of the Linux Netfilter package (see Chapter 2), and uses the connection-tracking capabilities at its core. NuFW provides authentication for each connection to a machine. It identifies new connections using the connection-tracking module, and so can differentiate between sessions, and multiple users from the same host or from behind a

common NAT device. This functionality may enhance security of most firewall systems that discriminate solely based on IP addresses.

While the suite can provide added security to a gateway, it does not provide any policy-configuration benefits to administrators. Neither does it appear to enhance network accessibility. Additionally, the NuFW suite does not work well with destination network address translation.

Chapter 7

Conclusion

This research has considered several issues regarding VPN access control and managing multiple resources on a network. As a solution to these problems, this research offers an access control framework that facilitates VPN user access as well as providing centralized management for network resources. At the same time this framework, called the Access Control Service (ACS), provides a means by which legacy access control systems network-wide can benefit from newer access control technology (such as decentralized delegated access) with minimal modification. As a proof of concept for the functionality of the framework, this research also presented a prototype that successfully facilitates VPN user access to resources using firewall and VPN access control. The prototype has also served to refine the model for the ACS framework.

A number of concerns about the current implementation have been raised in this paper, but for each of these the research presented here has offered a practical solution. Additionally, this work has identified ways of improving the framework while inviting more research on the identified issues of VPN access control, as well as the solution presented by the Access Control Service.

Appendix A

The Complete ACS Grammar

This appendix gives the ACS grammar currently in use by my implementation. The input may be a file or a message received on the mediators dedicated port. A message or file is considered 'valid' if it contains one or more object declarations at the top level (outside of any curly braces). The complete grammar includes syntax for activating or deactivating client and server objects (using UP or DOWN keywords after identifying the object).

The complete ACS grammar also enforces that a conn object declared within a client that is defined at the top level of an input must include an IP address assignment for the server on the other end of that connection.

In this grammar, a server object may include a *type* declaration. This allows an administrator to specify a template by which future *conn* objects can be created for clients authorized to access the particular resource (the resource corresponding to the server object).

```
<Input>
        <Declarations>

<Declarations>
        <Server Declaration> <Declarations>
        <Client Declaration Sans Server> <Declarations>
        <Server State Change> <Declarations>
```

<Client State Change> <Declarations>

NIL

<Server Declaration>

Server *uint*

Server *uint* **{**<Server Definition>**}**

<Server State Change>

Server *uint* **up**

Server *uint* **down**

<Client State Change>

Client *uint* **up**

Client *uint* **down**

<Client Declaration>

Client *uint*

Client *uint* **{**<Client Definition>**}**

<Client Declaration Sans Server>

Client *uint*

Client *uint* **{**<Client Definition Sans Server>**}**

<Server Definition>

<Client Declaration> <Server Definition>

<Host Declaration> <Server Definition>

Type *uint* **{**<Type Definition>**}** <Server Definition>

NIL

<Client Definition>

<Conn Declaration> <Client Definition>

<Host Declaration> <Client Definition>

NIL

<Client Definition Sans Server>

<Conn Declaration Sans Server> <Client Definition Sans Server>

<Host Declaration> <Client Definition Sans Server>

NIL

<Host Declaration>

Host *string*

<Conn Declaration>

Conn {<Conn Definition>**}**

<Conn Declaration Sans Server>

Conn {<Conn Definition Sans Server>**}**

<Conn Definition>

Type *uint* **{**<Type Definition>**}**

<Host Declaration> <Conn Definition>

<Conn Definition Sans Server>

<Host Declaration> <Conn Definition>

Glossary

26sec – The native IPSec module of the 2.6 family (2.6.x) of Linux kernels.

ATM – Stands for *Asynchronous Transfer Mode*. This is a network architecture that uses fixed-length units (53-byte frames) for data transmission. Because ATM frames are fixed-length they can be processed quickly. This allows ATM to be used for data transfer that where speed is a priority (e.g. Audio and Video data).

AH – Stands for Authentication Header. The AH protocol encrypts the payload and some fields in the header of a packet. It provides integrity for an IPSec packet.

Buffer-Overflow – When a computer program receives input it typically writes this in a buffer before working on it. Because buffers are fixed chunks of memory, a program that does not correctly check the size of its input can write beyond the end of a buffer and into other regions in memory. This scenario is called buffer-overflow. Well-crafted buffer-overflows can cause a program to write new instructions into its memory that can be executed later.

CHAP – Stands for *Challenge Handshake Authentication Protocol*. CHAP is a standard for verifying a user's identity much like PAP. Unlike PAP, CHAP uses encryption for the username and password by sending the user an encryption key before any further information is exchanged. CHAP may periodically request re-authentication to ensure that the user at the other end of the connection has not changed.

CORBA – Stands for *Common Object Request Broker Architecture*. CORBA is a specification for an architecture by which applications may communicate in a distributed computing environment.

Datagram – See *Frame*.

DSN – Stands for *Dedicated Secure Network*. This is a synonym for a leased line.

EAP – Stands for *Extensible Authentication Protocol*. EAP is a standard used for transmitting access requests and credentials between devices. EAP provides a framework by which one device may select certain security parameters (e.g. which encryption algorithm to use) for communicating with another device. It also supports the transmission of simple messages to indicate acceptance or rejection. For example, a wireless device may request network access from an Access Point using EAP, and receive a 'success' or 'failure' decision for the request. Similarly, using EAP, one server may forward an access request to a remote server for authentication and approval.

ESP – Stands for *Encapsulation Security Payload*. The ESP protocol offers encryption for the payload of an IPSec packet. It optionally offers integrity if the entire underlying IP packet is encrypted rather than just the transport-layer data (TCP, UDP, etc).

Eth0 – The name of given to one of the Ethernet network interface cards (NICs) in Linux systems. Other cards may be assigned eth1, eth2, etc.

Fedora – A distribution of Linux. See http://fedora.redhat.com.

Firewall – A firewall is a set of rules that a device uses to make decisions about packets based on network-layer information.

Frame – A unit of data transmitted across a network. Depending on the network protocol, a frame may also be called a 'packet' or a 'datagram'.

Frame Relay – A fast data transfer protocol. Frame Relay avoids accounting and error checking of data in favor of increased throughput. It is based on X.25 technology but is not restricted to analog circuitry.

Freeswan – An IPSec VPN server implementation for Linux. The project has been discontinued. Openswan and Strongswan have succeeded Freeswan. See http://www.freeswan.org

GUI – Stands for *Graphical User Interface*. It usually refers to all that part of an application that includes windows, buttons, etc, and responds to mouse clicks and key presses as input to the application.

HTML – Stands for *Hyper Text Markup Language*. HTML defines parameters (called tags) for formatting documents displayed in a web browser.

HTTP – Stands for *Hyper Text Transfer Protocol*. This is the standard used by the World Wide Web for requesting and transmitting documents over the Internet.

ICMP – Stands for *Internet Control Message Protocol*. ICMP is used for communicating errors and control messages. These control messages are typically information about the state of a host or a connection. Some common ICMP control messages are the *echo-request* and *echo-reply* messages. A host sends an ICMP echo-request to determine if another host is available on the network. The site that receives this message responds with an echo-reply in acknowledgment.

IETF – Stands for *Internet Engineering Task Force*. The IETF is an open international community of network users, organizations, and researchers that is responsible for the technical management of the Internet. The community publishes standards for protocols and best practices as RFCs.

IP – Stands for *Internet Protocol*. IP is a protocol that network-layer devices us to communicate. Under this protocol, a packet is divided into two main parts, a header and a payload. The header contains information such as the source and destination addresses of the packet, as well as the payload length. The payload is the actual data that the packet transmits.

IP Filter – This is a policy that makes decisions based on the header information of an IP packet.

IPSec – Stands for *Internet Protocol Security*. This describes a standard for tunneling and encrypting IP packets using two protocols, AH and ESP.

Ipsec0 – the name of the virtual interface assigned to the first IPSec tunnel created in a Linux system. It may also correspond to eth0. Subsequent tunnels may be assigned 'ipsec1', 'ipsec2', etc. Linux kernels using 26sec do not provide this interface.

Iptables – A Linux application for controlling the Kernel's routing mechanism. Iptables was created as part of the Netfilter project.

Kerberos – A network authentication protocol developed at the Massachusetts Institute of Technology (MIT). Under Kerberos, a user is authenticated by a trusted third-party and issued a ticket as proof of authentication. All servers that recognize the third party will honor this ticket.

KLIPS – Stands for ***K****erne****L*** ***I****nternet* ***P****rotocol* ***S****ecurity*. KLIPS is a module, distributed as part of FreeSwan, which provides kernel support for IPSec on Linux kernel versions earlier than 2.6.

L2F – Stands for *Layer 2 Forwarding*. L2F is a standard that Cisco Systems® developed for tunneling link-layer (layer 2) frames over a point-to-point connection.

L2TP – Stands for *Layer-2 Tunneling Protocol*. L2TP is an IETF standard that combines PPTP and L2F.

LAC – Stands for *Layer-2 Tunneling Protocol Access Concentrator*. As part of the L2TP standards, a LAC is a server that tunnels PPP datagrams to another server for authentication.

LAN – Stands for *Local Area Network*. A LAN is a cluster of devices that covers a small geographical area (such as a room or building) where communication between these devices is not done over the Internet.

Leased Line – Private physical network infrastructure that is managed by an ISP for its customers. This infrastructure usually comprises of cable, routers, and optical fiber.

LNS – Stands for *Layer-2 Tunneling Protocol Network Sever*. As part of the L2TP standards, the LNS receives tunneled PPP datagrams for authentication.

Masquerade – This is a form of Network Address Translation, also called 'many-to-one NAT,' where multiple devices with different IP addresses are made to appear as if sharing the same IP address.

MS-CHAP – Stands for *Microsoft Challenge Handshake Authentication Protocol*. MS-CHAP is Microsoft's implementation of CHAP.

NAT – Stands for *Network Address Translation*. This is a mapping of the fields of a packet header to another set of fields in which a subset of the source, destination, or port addresses of the original packet is changed.

Netfilter – An open source project for Linux that provides a framework for managing firewalls, network address translation, and packet mangling. See http://www.netfilter.org.

Network Address Translation – See *NAT*.

Network Interface Card – See *NIC*.

NIC – Stands for *Network Interface Card*. A NIC is a device that serves as a computer's interface to a network; it is the first part of the computer to receive packets from the network and the last part to handle packets before they are sent off onto the network.

Openswan – A successor of Freeswan. A company called Xelerance actively maintains Openswan. See http://www.openswan.org

OpenVPN – An SSL VPN server implementation for Linux, Windows and Mac OS X. See http://openvpn.net

Packet – See *Frame*.

PAM – Stands for *Pluggable Authentication Modules*. PAM is an authentication system that controls access to Linux.

PAP – Stands for *Password Authentication Protection*. PAP is a method of verifying a user's identity by transmitting a username and password over the Internet to be compared with a stored username and password on another machine. The username and password is transmitted unencrypted.

PKI – Stands for *Public Key Infrastructure*. It is a system that defines a number of entities (end-users, principals, trusted authorities, and issuers), and provides a framework for identifying these entities, maintaining confidentiality, and verifying the integrity of data.

End-User –	The entity that owns a key-pair (public and private key), and a certificate containing this public key.
Principal –	The entity that signs some data with a key.
Trusted Authority –	The entity that endorses an end-user's certificate.
Issuer –	The entity that gives an end user a certificate.

The system uses structures such as certificates, and revocation lists as part of this framework.

POP – Stands for *Point of Presence*. It is the first device (switch, AP, router, etc) on a network to which a user connects even before authenticating with the network's gateway.

PPP – Stands for *Point-to-Point Protocol*. PPP is a standard for creating and running network protocols over a dial-up connection (e.g. modem cable, telephones lines or any other serial links).

PPTP – Stands for *Point-to-Point Tunneling Protocol*. PPTP is a tunneling standard, invented by Microsoft®, which encapsulates PPP in other networking protocols such as IP.

Process – If a computer program is a set of instructions that a computer can execute then a process is an execution of that program. As this definition suggests, a computer can usually create many processes from one program.

Python – An open source object-oriented interpreted language. Python can be interpreted in most of the common platforms.

RADIUS – Stands for *Remote Authentication Dial In User Service*. RADIUS is an authentication, accounting, and authorization protocol for network access.

RFC – Stands for *Request For Comments*. The RFCs are a series of documents covering issues concerning the Internet and its usage. The IETF is responsible for gathering and published these documents.

SA – Stands for *Security Association*. A security association is a relationship between two or more entities (i.e. devices, processes, etc) that defines how each entity will use

security parameters (e.g. number of bits for encryption) and services (i.e. encryption, hashing, and padding algorithms) to communicate securely.

SDSI/SPKI – Stands for *Simple Distributed Security Infrastructure* / *Simple Public Key Infrastructure*. SDSI/SPKI is a certificate standard that was developed to overcome some of the shortcomings of X.509. The SDSK/SPKI standard includes support for delegation and certificate chains.

SSL – Stands for *Secure Socket Layer*. The SSL protocol defines a method for authenticating one end of a connection (or optionally both), establishing a number of shared secrets, and encrypting data between both end-points for the duration of the connection.

Strongswan – A successor to Freeswan. The Strongswan project differs from the Openswan project in its goals; Strongswan focuses more on using strong security authenticating IPSec connections. For this reason Strongswan provides more sophisticated PKI capabilities than Openswan. See http://strongswan.org.

TCP – Stands for *Transmission Control Protocol*. The protocol provides a means for guaranteeing that IP packets are not only delivered, but that these packets are received in the correct order. The TCP header also allows the specification of source and destination ports.

TCB – Stands for *Trusted Computing Base*. The TCB includes all components of a system that have policies for securing sensitive information that the system uses.

TLS – Stands for *Transport Layer Security*. The TLS protocol is the IETF's standard for SSL.

UDP – Stands for *User Datagram Protocol*. Unlike TCP, UDP does not provide confirmation on the receipt of an IP packet. Neither does it guarantee that packets are

received in the correct order. Like TCP, UDP also provides port specification in its header.

VLAN – Stands for *Virtual Local Area Network*. A VLAN is much like a LAN with the exception that the devices on the network are not necessarily in proximity with one another (i.e. there are no geographical constraints).

X.25 – A data transmission protocol for analog circuitry. The Consultative Committee on International Telephone and Telegraph (CCITT) developed the X.25 standard.

X.509 - A digital certificates standard developed by the International Telecommunications Union (ITU).

X server – This is the part of the X Windowing System that interacts with the graphics hardware to draw windows and receive input from the user on behalf of an application.

X Windowing System – This is a windowing system found on most modern operating systems. It is used for bitmap display by interfacing with the graphics hardware on behalf of applications. The system is commonly used over networks with an application running on one machine but its graphical output appearing on another machine.

Bibliography

[AMW01] Kevin Atkinson, Sinisa Milivojevic, Michael Widenious. A C++ API for MySQL. http://dev.mysql.com/doc/plusplus/en. Retrieved 05/17/2005.

[And03] Oskar Andreasson. Iptables Tutorial 1.1.19. http://iptables-tutorial.frozentux.net. 2001 - 2003. Retrieved 04/23/2005.

[BV98] Larry J. Blunk and John R. Vollbrecht. PPP Extensible Authentication Protocol (EAP). IETF RFC 2284, March 1998.

[CB97] Tzi-Cker Chiueh, Allen Ballman. Performance optimization of Internet firewalls. In Video Techniques and Software for Full-Service Networks. Vol. 2915, p. 168-173. Copyright 1997 SPIE--The International Society for Optical Engineering.

[CBR03] William R. Cheswick, Steven M. Bellovin, Aviel D. Rubin. Firewall and Internet Security (2nd Ed): Repelling the Wily Hacker. Addison-Wesley professional Computing Series, Addison- Wesley, Copyright 2003 AT&T and Lumeta Corporation.

[DeL02] Jacco de Leeuw. Using a Linux L2TP/IPsec VPN server. http://www.jacco2.dds.nl/networking/freeswan-l2tp.html, 2002. Retrieved 05/01/2005.

[EFL+99] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI Certificate Theory. IETF RFC 2693, September 1999.

[FKT01] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 15(3), 2001.

[FKT97] I. Foster, C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications, 11(2):115-128, 1997.

[Fos02] I. Foster. What is the Grid? A Three Point Checklist. GRIDToday, July 20, 2002.

[Fra01] Moye Fraser, Understanding Virtual Private Networks, Sans Institute, March 2001.

[FreeR] The Free RADIUS Project. http://www.freeradius.org. Retrieved 05/01/2005.

[Ful05] Jim Fulton. MIT X Consortium. XAUTH MAN PAGES. http://www.xfree86.org/4.4.0/xauth.1.html. Retrieved 04/11/2005.

[Glob05] University of Chicago. The Globus Alliance. Retrieved 04/2/2005.

[Goff04] Nicholas C. Goffee, Greenpass Client Tools for Delegated Authorization in Wireless Networks, July 2004, Dartmouth College.

[HC98] D. Harkins, D. Carrel. The Internet Key Exchange (IKE). Request for Comments (RFC) Standards Track 2408. November 1998.

[Hos04] Charlie Hosner. OpenVPN and the SSL VPN Revolution, Sans Institute, March 2004.

[iDFNS05] iDEFENSE Security Advisory 01.26.05. http://www.idefense.com/application/poi/display?id=190&type=vulnerabilities&flashstatus=true. Retrieved 04/11/2005.

[ISS] Securing Database Servers. ISS white paper. URL: documents.iss.net/whitepapers/securedbs.pdf. Retrieved 05/27/2005

[KA98a] S. Kent and R. Atkinson. IP authentication header, November 1998. RFC 2402.

[KA98b] S. Kent and R. Atkinson. IP encapsulating security payload (ESP), November 1998. RFC 2406.

[Kim04] Sung Hoon Kim. Greenpass RADIUS Tools for Delegated Authorization in Wireless Networks. Master's thesis, Dartmouth College, June 2004.

[KN93] John Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (Version 5). Internet Request for Comments RFC-1510. September 1993

[KPS02] Charlie Kaufman, Radia Perlman, Mike Speciner, "Network Security: PRIVATE Communication in a PUBLIC World", Prentice Hall Series in Computer Networking and Distributed Systems, Prentice Hall 2002.

[LBTMO] Lobotomo Software. Ipsecuritas 2.0.6. http://www.lobotomo.com/products/IPSecuritas. Retrieved 05/01/2005.

[LDOC] Point-to-Point Protocol - HOWTO. From the Linux Documentation Project. www.tldp.org. Retrieved 04/12/2005.

[MSFT00] Microsoft Corporation, Configuring Remote Access/Virtual Private Networking, April 2000.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9): 33-38. September 1994.

[NTFLTR] The Netfilter/IPTables Project. http://www.netfilter.org. Retrieved 04/01/2005.

[NuFW] Now User Filtering Works. http://www.nufw.org. Retrieved 05/04/2005.

[OpenSSL] The Open Source toolkit for SSL/TLS. http://www.openssl.org. Retrieved 04/7/2005

[Open+05] Open Source Development Corporation, Astaro Network Security, and Missions. Integrated Secure Communications System Project Page. http://iscs.sourceforge.net/. Retrieved 05/2/2005.

[PCWEBa] What is packet filtering? - A Word Definition From the Webopedia Computer Dictionary. http://www.pcwebopedia.com/TERM/S/stateful_inspection.html. Retrieved 04/07/2005.

[PCWEBb] What is stateful inspection? - A Word Definition From the Webopedia Computer Dictionary. http://www.pcwebopedia.com/TERM/S/stateful_inspection.html. Retrieved 04/07/2005.

[Pow04] Kimberly Powell. Testing the Greenpass Wireless Security System. Senior honors thesis, Dartmouth College, June 2004.

[Ran03] David A. Ranch. Linux IP Masquerade HOWTO. http://en.tldp.org/HOWTO/IP-Masquerade-HOWTO, Nov 2003. Retrieved 04/25/2005.

[RHLa] Red Hat Inc. GNOME LOKKIT. http://www.redhat.com/docs/manuals/linux/RHL-8.0-Manual/custom-guide/s1-basic-firewall-gnomelokkit.html. Retrieved 05/01/2005.

[RHLb] Glossary. Red Hat Linux 6.2 Manual. www.redhat.com/docs/manuals/linux/RHL-6.2-Manual/getting-started-guide/ch-glossary.html Retrieved 04/11/2005.

[Rig00] Carl Rigney, RADIUS Accounting. IETF RFC 2866, June 2000.

[RMKGL96] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear. Address Allocation for Private Internets. Request For Comments (RFC) Best Current Practice. February 1996.

[RS02] Rodrigo Blanco Rincon, Gunter Schafer, On Securing Wireless LANs and Supporting Nomadic Users with Microsoft's IPSec Implementation, 2002.

[Russ02] Rusty Russel. Linux 2.4 Packet Filter Howto. www.netfilter.org, 2002. Retrieved 05/01/2005.

[RWC00] Carl Rigney, Ward Willats, and Pat R Calhoun. RADIUS Extensions. IETF RFC 2869, June 2000.

[RWRS00] Carl Rigney, Steve Willens, Allan C Rubens, and William Allen Simpson. Remote Authentication Dial In User Service (RADIUS). IETF RFC 2865, June 2000.

[SBDG02] Marc-Alain Steinemann, Torsten Braun, Marc Danzeisen, Manuel Gunther, Virtual Private Networks**,** Wiley Encyclopedia of Telecommunications, 2002, ISBN 0-471-36972-1, pp. 2807-2815.

[SGK+04a] Sean Smith, Nicholas C. Goffee, Sung Hoon Kim, Punch Taylor, Meiyuan Zhao, and John Marchesini. Greenpass: Flexible and Scalable Authorization for Wireless Networks. Technical Report TR2004-484, Dartmouth College, 2004.

[SGK+04b] Nicholas C. Goffee, Sung Hoon Kim, Sean Smith, Punch Taylor, Meiyuan Zhao, and John Marchesini. Greenpass: Decentralized, PKI-based Authorization for Wireless LANs. In *3rd Annual PKI Research and Development Workshop*, 2004.

[Sull04] John A. Sullivan III. The Brief How and Why of ISCS. Nexus Management. http://iscs.sourceforge.net/HowWhyBrief.html. Copyright 2003-2004. Retrieved 04/2/2005.

[SWANa] FreeS/WAN Project. http://www.freeswan.org. Retrieved 05/17/2005

[SWANb] Openswan. http://www.openswan.org. Retrieved 05/17/2005

[SWANc] Strongswan – IPSec for Linux. http://www.strongswan.org. Retrieved 05/17/2005.

[SWANd] XAUTH - authenticator. http://wiki.openswan.org/index.php/XAUTH%20authenticator. Retrieved 04/11/2005.

[TECHWEB] TechWeb Encyclopedia. http://www.techweb.com/encyclopedia/defineterm.jhtml?term=x+server. Retrieved 04/11/2005.

[TOB98] DJ Thomsen, D O'Brien, J Bogle. Role-Based Access Control Frameworks for Network Enterprises, 1998.

[Vep02] Linas Vepstas. Linux Network Address Translation. http://linas.org/linux/load.html. Retrieved 05/01/2005.

[VPN04] VPN Consortium. VPN Technologies: Definitions and Requirements (white paper). http://www.vpnc.org/vpn-technologies.html, July 2004. Retrieved 05/10/2005.

[WFK+04] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. Globus Organization. 3rd Annual PKI R&D Workshop, 2004.

[Wob97] John Wobus. Basic Glossary on Campus Networks, http://jhunix.hcf.jhu.edu/~tnaugler/770.512/Common_files/LANs/Wobus/lan-glossary.html. March 1997

[Yon04] James Yonan.http://openvpn.net. Retrieved 04/ 06/2005.

[ZRT95] G. Ziemba, D. Reed, P. Traina. Security Considerations for IP Fragment Filtering, Internet Networking Group, RFC 1858 Informational, October 1995.