

University of Montana

## ScholarWorks at University of Montana

---

Graduate Student Theses, Dissertations, &  
Professional Papers

Graduate School

---

1996

### Network management and security for Unix

Doddy Vamsi Bathanaiah  
*The University of Montana*

Follow this and additional works at: <https://scholarworks.umt.edu/etd>

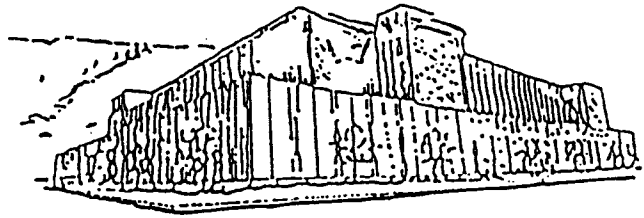
**Let us know how access to this document benefits you.**

---

#### Recommended Citation

Bathanaiah, Doddy Vamsi, "Network management and security for Unix" (1996). *Graduate Student Theses, Dissertations, & Professional Papers*. 3074.  
<https://scholarworks.umt.edu/etd/3074>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact [scholarworks@mso.umt.edu](mailto:scholarworks@mso.umt.edu).



Maureen and Mike  
**MANSFIELD LIBRARY**

The University of **MONTANA**

---

Permission is granted by the author to reproduce this material in its entirety,  
provided that this material is used for scholarly purposes and is properly cited in  
published works and reports.

*\*\* Please check "Yes" or "No" and provide signature \*\**

Yes, I grant permission   X    
No, I do not grant permission       

Author's Signature   Maureen and Mike  

Date   2/9/96  

Any copying for commercial purposes or financial gain may be undertaken only with  
the author's explicit consent.



# Network Management and Security for Unix

by

Doddy Vamsi Bathanaiah

B.Tech. Jawaharlal Nehru Tech University, India. 1993

presented in partial fulfillment of the requirements

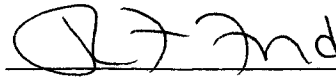
for the degree of

Master of Science

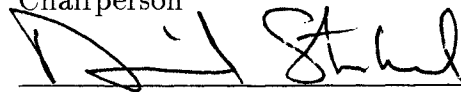
The University of Montana

February 1996

Approved by:



Chairperson



Dean, Graduate School

MARCH 1, 1996

Date

UMI Number: EP36238

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI EP36238

Published by ProQuest LLC (2012). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

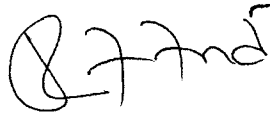
All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 - 1346

Network Management and Security for Unix (63 pp.)

Director: Ray Ford

A handwritten signature in black ink, appearing to read 'Ray Ford', with a stylized, cursive script.

Computer networks have revolutionized our use of computers. As the computer networks developed vastly, the need for their management and security became nothing but essential. To deal with the multivendor environment a network management system is needed that is based on standardized network management protocols and applications. The users should be allowed access to the services only after they are authorized and authenticated to do so. This paper investigates how to manage the UNIX networks using SNMP and also how to deal with security of UNIX machines using Kerberos and firewall. It also discusses about IP spoofing and hijacking and some design decisions to take in order to avoid those attacks.

## Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.	Introduction . . . . .	1
<b>2</b>	<b>Linux</b>	<b>7</b>
1.	Introduction . . . . .	7
2.	Summary . . . . .	11
<b>3</b>	<b>Simple Network Management Protocols - Version 1 and 2</b>	<b>14</b>
1.	Introduction . . . . .	14
2.	Network Management Architecture . . . . .	15
3.	MIB structure . . . . .	20
4.	SNMP Formats . . . . .	22
5.	Transmission of an SNMP message . . . . .	23
6.	Receipt of an SNMP message . . . . .	24
7.	Overview of SNMPv1 . . . . .	24
7.1.	Limitations of SNMPv1 . . . . .	26
8.	Overview of SNMPv2 . . . . .	26
8.1.	GetRequestPDU . . . . .	27
8.2.	GetNextRequestPDU . . . . .	28
8.3.	GetBulkRequestPDU . . . . .	29
8.4.	SetRequest PDU . . . . .	30
8.5.	SNMPv2 Trap PDU . . . . .	30
8.6.	InformRequest PDU . . . . .	31
9.	SNMP Installation under Linux . . . . .	32
10.	Summary . . . . .	34

<b>4</b>	<b>Kerberos</b>	<b>35</b>
1.	Introduction . . . . .	35
2.	Kerberos Authentication and Security . . . . .	36
3.	How Kerberos works . . . . .	37
4.	Kerberos Encryption . . . . .	38
5.	The Kerberos Ticket . . . . .	38
6.	Application request and response . . . . .	39
7.	Obtaining additional tickets . . . . .	42
8.	Protecting application data . . . . .	43
9.	Infrastructure and Cross-Realm Authentication . . . . .	44
10.	Other approaches for improving security . . . . .	45
11.	One-time passcodes . . . . .	45
12.	Public-key Cryptography . . . . .	46
13.	Limitations of Kerberos . . . . .	47
14.	Kerberos Installation under Linux . . . . .	48
15.	Summary . . . . .	50
<b>5</b>	<b>Firewalls</b>	<b>51</b>
1.	Introduction . . . . .	51
2.	What can a firewall protect against ? . . . . .	52
3.	What can't a firewall protect against ? . . . . .	53
4.	Design Decisions . . . . .	54
5.	About Proxy Servers . . . . .	55
6.	Internal Components of a Firewall . . . . .	55
7.	Typical Screened Host Gateway . . . . .	56
8.	Typical Screened Subnet . . . . .	58



9.	IP spoofing . . . . .	59
10.	Hijacking tools . . . . .	60
11.	Solutions . . . . .	61
12.	Summary . . . . .	63

# List of Figures

3.1	Protocol context of SNMP . . . . .	18
3.2	MIB2 Object Groups . . . . .	21
3.3	SNMP Formats . . . . .	23
5.1	Typical Screened Host Gateway . . . . .	57
5.2	Typical Screened Subnet . . . . .	58

# Chapter 1

## Introduction

### 1. Introduction

Linux is a freely distributable UNIX clone for personal computers. It was originally developed by Linus Torvalds[1]. The system now consists of parts contributed by dozens of individual developers. It currently runs on Intel 386, 486 and Pentium machines and it is now being ported to other architectures such as Motorola 680x0, DEC/Alpha and MIPS processors. Linux can coexist with other operating systems such as MS-DOS, OS/2 and so forth. Linux uses its own drive partitions and therefore does not interfere with other operating systems in anyway. The Linux boot manager is used to select which operating system to load at the boot time.

The idea of networking is as old as telecommunications itself. A network is a collection of hosts that are able to communicate with each other by relying on the services of a number of hosts that relay data between the participants. Network communication is impossible without some sort of standard language or code. In computer networks these are collectively referred to as *protocols*. The two important protocols for the work described here are TCP and UDP. TCP, for *transmission control protocol*, is a

connection oriented protocol meaning that participants should establish connection before transmitting. This provides reliable, full duplex, stream service on which many application protocols depend. In contrast UDP, for *user datagram protocol*, is a connectionless protocol meaning that there is no actual connection between participants before transmitting. This provides unreliable, full duplex service[5].

A port is an attachment point for a network connection, used by a transport protocol to distinguish among multiple destinations within a computer. Ports are used as for network connections. If an application running on a host wants to offer a certain service over the network, it attaches itself to a port and waits for clients, in the form of service requests from other hosts. A client that wants to use a service allocates a port on its local host and connects to the server's port on the remote machine. An important property of a port is that once a connection has been established between the client and the server, the port becomes free so that another copy of the server may attach to the server port and listen for more clients. This permits, for instance, several concurrent *telnet* service requests to be active on the same host, all connected using the same port. Both TCP and UDP *name* their ports by numbers from 7 to 2401. Though they use the same numbering scheme, these numbers do not conflict. TCP port 513, for example, is different from UDP port 513. In fact, these ports serve as access points for two different commonly offered services, namely *rlogin*(TCP) and *rwho*(UDP)[4].

As more and more machines are networked together the need for network management and network security becomes increasingly important. The most commonly used mechanism for network management is SNMP, which stands for Simple Network Management Protocol[8]. The most important network security mechanism used in

Unix systems is Kerberos[2]. A brief overview of these facilities follows.

The components of SNMP include a network management system (NMS), network agents, protocol data units, the management information base (MIB) and the structure of management information (SMI). The NMS monitors and controls the SNMP agents found within devices. It provides information to a human network manager who uses the machine to issue commands and interpret statistical information gathered from a network. Network agents are devices such as routers or bridges that are to be managed on an SNMP network. These agents monitor network information such as the number of connections, packets transmitted, or error conditions at their specific location. The agents provide this information to network management stations, which use the User Datagram Protocol to communicate with them. Agents and managers communicate with protocol data units. Generally all managers send commands and receive responses from the agents. The management information base (MIB) is a repository that contains information about network objects, such as modems and routers, and the management attributes of these objects. The structure of management information (SMI) is the rule book for SNMP's management information base. Its function is similar to that of a set of grammar rules that define the way to build sentences. The SMI defines how management protocols get information about the objects which represent the network devices. The SNMP agents and stations communicate through standard messages, where each message is a single packet exchange. Because of this, SNMP is implemented using UDP as the layer 4, or transport layer, protocol according to William Stallings[8]. UDP uses connectionless service, so SNMP does not have to maintain a connection between an agent and station to transmit a message.

Kerberos is a trusted third party authentication service based on the model developed by Needham and Schroeder[10]. This project was started at M.I.T in 1983 and its goal was to create a network security environment. It is now in its fifth version. The basic Kerberos protocol contains a client and a collection of servers. The version 5 implementation of Kerberos contains three types of servers and one type of client. The client program prompts the user for login requests and passwords. The three servers are the authentication server, ticket granting Server (TGS) and the actual service provider, such as FTP. The authentication server checks whether the user exists in a database. If the user exists the server sends a message with a *ticket* which gives permission for the client to communicate with the TGS. The client then sends a message to the TGS which includes its *ticket*. If the ticket is accepted, the TGS returns a message to the client with a different ticket to an end-server. This ticket grants the client access to that particular end-server and so the client starts communicating with the end-server using that ticket. Kerberos also creates temporary secret keys, called session keys, which are given to client and server and no one else. A session key is used to encrypt messages between the two parties, during the session. After the session, the session key is destroyed.

Thus, Kerberos provides three different levels of protection. It provides authentication at the beginning of a network connection, after which all further communications are assumed to come from the authenticated entity. It also provides authentication for each message sent from one entity to another. Finally, it provides both authentication and encryption for each message sent from one entity to another.

A *firewall* is a special host that sits between the outside network and internal network[19]. This host does not send any information regarding routing within the internal network

to the outside, thereby making the internal network invisible to the outside world. The firewall machine should not mount any file systems via NFS or RFS. Password security must be rigidly enforced on a firewall. Only users who require access to outside networks and users who must log into the system should have accounts on the machine. Anonymous FTP and other similar services should only be provided by the firewall machine. The */etc/ftpusers* and */etc/shells* files are used to restrict the access. Services that are not needed should be disabled in */etc/inetd.conf*. The *syslog* on the firewall machine is configured to log all information to a remote host which prevents an intruder from deleting information from the log files. Compilers and loaders should be deleted from the firewall machine. This will prevent hackers from compiling programs there thereby stopping programs like Internet worm. The main purpose of the firewall is to prevent attackers from accessing other hosts in the internal network.

The project described here involves the application of SNMP, Kerberos and firewall facilities in Linux. Also I include comments on DES, Data Encryption Standard, which is a standard encryption procedure used in Kerberos.

Briefly summarized, the work involved in this project includes:

- . Installation of Linux on a IBM compatible personal computer
- . Installation of SNMP under Linux, by porting a public domain SNMP implementation
- . Installation of Kerberos under Linux, by porting a public domain Kerberos implementation
- . Analysis of the use of firewalls in the resulting system to prevent spoofing attacks

from being succesful.



# Chapter 2

## Linux

### 1. Introduction

Linux was originally developed by Linus Torvalds. Linux is a complete UNIX capable of running X windows, TCP/IP, Emacs, UUCP, mail and lots of other unix utilities. Almost all of the major free software packages are ported to Linux. Linux is a complete multitasking, multiuser operating system meaning that many users can be logged into the same machine at the same time running multiple programs simultaneously. Linux supports virtual consoles which allows the user to switch between multiple login sessions from the system console in text mode. The kernel can emulate 387 FPU instructions so that the systems without a math coprocessor can run programs that require floating point math instructions[4].

Linux currently supports systems with an Intel 80386, 486 and Pentium processors. This includes all variations on this CPU type such as the 386SX, 486SX and 486DX. The system motherboard must use ISA or EISA bus architecture but not MCA architecture.

Linux requires very little memory to run compared to other operating systems. A system with only 2 MB of memory can run the Linux operating system. Linux supports all IDE, ESDI and RLL controllers. It also supports a number of popular SCSI drive controllers although the support for SCSI is more limited because of wide range of controller interface standards. SCSI controllers like Adaptec AHA1542B, AHA1742A, AHA1740, TMC850, TMC950, Seagate ST02 are all supported by Linux. Linux supports all standard Hercules CGA, EGA, VGA and Super VGA video cards and monitors for the default text based interface. CDROM drives like Sony CDU541, Mitsumi are all supported by Linux. Linux supports the standard ISO 9660 filesystem for CDROMs. Linux supports a complete range of parallel printers and modems[4].

Linux supports various filesystem types for storing data. The most widely used filesystems are Minix, Xenix, ext2fs, ISO9660 and MS-DOS. The MS-DOS filesystem allows the user to access MS-DOS files directly by mounting an MS-DOS partition or floppy. Linux provides a complete implementation of TCP/IP networking which includes device drivers for many popular ethernet cards, SLIP, PLIP, PPP and NFS. Almost all of TCP/IP services are supported.

The Linux kernel uses the special protected mode features of the Intel 80x86 and Pentium processors. Linux supports demand-paged-loaded executables. That is only those segments of a program which are actually used are read into memory from disk. Also copy-on-write pages are shared among executables, meaning that if several instances of a program are running at once then they will share pages in physical memory, thereby reducing overall memory usage. In order to increase the amount of available memory, Linux also implements disk paging. When the system requires more physical memory it will swap out inactive pages to disk. Executables use dy-

namically linked shared libraries, meaning that executables share common library code in a single library file found on disk. This allows the executable files to occupy less space on disk, especially those that use many library functions. Also debugging facilities are provided in Linux environment.

Linux provides a complete UNIX programming environment including all of the standard libraries, programming tools, compilers and debuggers. Besides C and C++ many other compiled and interpreted programming languages have been ported to Linux, such as Smalltalk, Fortran, LISP and Scheme. Also scripting languages like Perl and Tcl/Tk are ported to Linux. The X window system is the standard graphics interface for UNIX machines, and standard X applications like xterm and xclock are supported. The only major caveats with X windows are the hardware and memory requirements. A 386 with 4 MB of RAM is capable of running X windows but 8 MB or more RAM is needed to run it efficiently.

Most TCP/IP networks use ethernet as the physical network transport. Linux supports many popular ethernet cards and interfaces for personal computers. Linux also supports SLIP (Serial Line Internet Protocol) which allows the users without ethernet to connect to the Internet via modem. UUCP (Unix to Unix Copy) is an older mechanism used to transfer files, electronic mail and news between UNIX machines. If there is no SLIP access then the Linux system can be configured to work with UUCP. Many popular ethernet cards like 3com, ec503, Novell NE1000, Novel NE2000, WesternDigital WD8003 are supported by Linux.

Linux systems are structured like traditional Unix systems. *rc* files are system wide configuration scripts executed at boot time by *init* which starts up all of the basic sys-

tem daemons such as *cron*, *sendmail*. *init* also configures things such as the network parameters, system hostname and so on. *rc* files are usually found in the directory */etc/rc.d*[1]. Usually there are two files in this directory. One of them configures the network while the other starts various servers used by the TCP/IP suite.

All Linux operating systems run a super server that creates sockets for a number of services and listens on all of them simultaneously. When a remote host requests one of the services, the super server notices it and spawns the server specified for that port. The commonly used super server is *inetd* which is started at the boot time. Linux also supports NIS, the Network Information System. NIS provides generic database access facilities that can be used to distribute information such as that contained in the *passwd* and *groups* files to all hosts on your network which makes the network appear as a single system.

SLIP and PPP are widely used protocols for sending IP packets over a serial link. No hardware modifications are necessary to run SLIP or PPP because any serial port can be used. Apart from a modem and a serial board equipped with a FIFO buffer no hardware is needed. Just like SLIP, PPP is a protocol to send datagrams across a serial connection. PPP overcomes some of the deficiencies of SLIP. It lets the communicating sides negotiate options such as the IP address and the maximum datagram size at start up time and also it provides client authorization. Linux also supports NFS which is the most prominent network service that uses RPC. NFS allows the users to access files on remote hosts in exactly the same way as local files.

## 2. Summary

Linux can be installed on a PC either through CDROM or floppies. Briefly there are five steps in installing Linux: disk partitioning, booting from Linux installation media, creating Linux partitions, creating filesystems and swap space, and installing the software on the new filesystems.

1. *Partition the Hard drive:* The hard drive must be partitioned to create space for Linux. Usually other operating systems like DOS exist on the hard drive before Linux is installed. The partitioning process allows Linux and other operating systems to co-exist and hence allows users to have access to all operating systems.
2. *Linux installation boot media:* Each distribution of Linux has some kind of installation media (usually a boot floppy) to start installing the software. There are many kinds of boot diskettes. A specific boot diskette must be selected to match the hardware configuration of the machine, particularly the type of hard disk and the CD driver. The boot diskette is made off-line (typically under DOS) by copying the appropriate parts and drivers using a special program called “rawrite”. If the diskette is made using normal DOS “copy”, it will not work.
3. *Create Linux partitions:* After repartitioning to allocate space for Linux, Linux partitions are created on the empty part of the hard drive. The best way to do this is to use the Linux “fdisk” program. Device drivers are found in the directory “/dev” and they are used to communicate with devices on the system. Two partitions are created, one of *Linux native* and the other for *Linux swap* partition. Usually all the executables and libraries for Linux are installed in Linux native partition. The ideal

space needed for Linux is around 150 MB. As noted above, the Linux partitions can co-exist with DOS/Windows partitions that were “saved” on the disk when the hard drive was set up.

4. *Create filesystems and swap space:* One or more filesystems must be created to store files on the newly created partitions. There are several types of filesystems for Linux. Each filesystem type has its own format and set of characteristics such as filename length, maximum file size and so on. The most commonly used filesystem type is “Second Extended File System” or “ext2fs”. After the filesystems are created, a swap space is created, then the filesystems are enabled.

5. *Install the software and configure network:* Finally, Linux must be installed on the newly created filesystems using a installing program called “setup”. Prior to installation the source device (like floppies or CDs) must be selected. Then the software components to be installed must be selected and installed. After installing Linux, the network must be configured. An IP address must be given for the machine, along with a hardware network connection. The completed Linux system can be booted using LILO (Linux Loader), located in the directory */etc/lilo*. Prior to booting Linux, the *lilo* configuration file */etc/lilo/config* must be edited to make changes appropriate to the local configuration. Then *lilo* can be executed.

As noted above, multiple operating systems can co-exist on the hard disk. The first operating system listed in the configuration file becomes the default operating system. That is, if Linux is listed first, it is the default operating system that comes up when the machine is powered on. However, during booting process other installed operating systems can be selected to boot instead by typing their names at the appropriate

prompt. The detailed description of this process is given in Appendix A.

# Chapter 3

## Simple Network Management Protocols - Version 1 and 2

### 1. Introduction

Many network management protocols have been implemented over the years. ICMP, for *Internet Control Management Protocol*, is one which was used extensively in the early days of networking. On Internet Protocol data networks, network engineers used ICMP *Echo* and *Echo Reply* messages to gather limited information useful for network management[6]. Originally intended for sending control messages information between two network devices, most ICMP messages are not easily interpreted by humans. These messages work as follows. A host device on a network that receives an ICMP *Echo* must return an ICMP *Echo Reply* to the source host. A reply not received by the sending host can, but not always, indicate a lack of network connectivity between the two hosts. For example, an application called *ping* (Packet InterNet Groper) tests network connectivity to a remote device by sending an ICMP *Echo* to the device, then waiting for the ICMP *Echo Reply*. Unfortunately the echo/reply protocol has lot of drawbacks, such as unreliable delivery, need for polling and lim-



ited information return. Because of these drawbacks a standard network management protocol offering a better set of services was needed. Consequently, Simple Network Management Protocol, SNMP, was developed. SNMP is now the most widely used network management protocol. SNMP was developed to provide a basic, no-frills service for TCP/IP based environments. TCP/IP now dominates the interoperable communications software market, so SNMP dominates the interoperable network-management software market.

In order to manage today's systems and networks effectively, and to plan intelligently for the future of network management systems, the system manager needs an understanding of the general technology of network management and a thorough grasp of the details of the existing and evolving standards. SNMPv2 is a new proposed version of SNMP intended to remove many deficiencies of the original SNMP and to broaden its applicability to include OSI-based networks as well as TCP/IP-based networks. In this section we review the general network management architecture and facilities in the original SNMP. We discuss problems and deficiencies in SNMP, then describe SNMPv2 and how this proposed version addresses the existing deficiencies. Finally, we discuss the process of porting a version of SNMP to a Linux host.

## **2. Network Management Architecture**

The model of network management that is used for TCP/IP network management includes as the key elements a management station, a management agent, a management information base (MIB), a network management protocol, and a structure of management information[8].

The *management station* is typically a stand-alone device but may be a capability implemented on a shared system. In either case, the management station serves as the interface for the human network manager in the network management system. The management station will have, at minimum, a set of management applications for data analysis and fault recovery, an interface by which the network manager can monitor and control the network, the capability of translating the network manager's requirements into the actual monitoring and control of remote elements in the network, and a database of information extracted from the MIBs of all the managed entities in the network.

The other active element in the network management system is the *management agent*. Hardware platforms such as hosts, bridges, routers and hubs can be equipped with SNMP, to allow them to act as management agents. A management agent responds to requests for information and requests for actions from the management station and can asynchronously provide the management station with important information. A management station monitors the network by retrieving the values of MIB objects. A management station can cause an action to take place at an agent or can change an agent's configuration settings by modifying the value of specific variables.

The resources of the network are represented as objects. Each object is, essentially, a data variable that represents one aspect of the managed agent. The collection of objects is referred to as a *management information base*. The MIB functions as a collection of access points at the agent for the management station. These objects are standardized across systems of a particular class (e.g., bridges all support the same management objects).

The management station and agents are linked by the *network management protocol*. The protocol used for the management of TCP/IP networks is the simple network management protocol, SNMP. There are no specific guidelines in the standards as to the number of management stations or the ratio of management stations to agents. In general, it is prudent to have at least two systems capable of performing management station functions, to provide redundancy in case of failure. An other issue is the practical one of how many agents a single management station can handle. As long as SNMP remains relatively simple, that number can be quite high, certainly in the hundreds.

SNMP is designed to be an application level protocol that is a part of the TCP/IP protocol suite. It is intended to operate over the user datagram protocol (UDP) as illustrated in Fig 3.1[8]. For a stand-alone management station, a manager process controls access to the central MIB at the management station and provides an interface to the network manager. The manager process achieves network management by using SNMP, which is implemented on top of the layers: UDP, IP, and the relevant physical network control protocols for ethernet, FDDI, etc. Thus, each agent must implement SNMP, UDP and IP. In addition, there is an agent process that interprets the SNMP messages and controls the agent's MIB. For an agent device that supports applications that are based on TCP protocols, TCP must also be supported on the agent. The following figure provides a somewhat closer look at the protocol context of SNMP.

From a management station, four types of SNMP messages can be issued on behalf of a management application: GetRequest, GetNextRequest, GetBulkRequest and

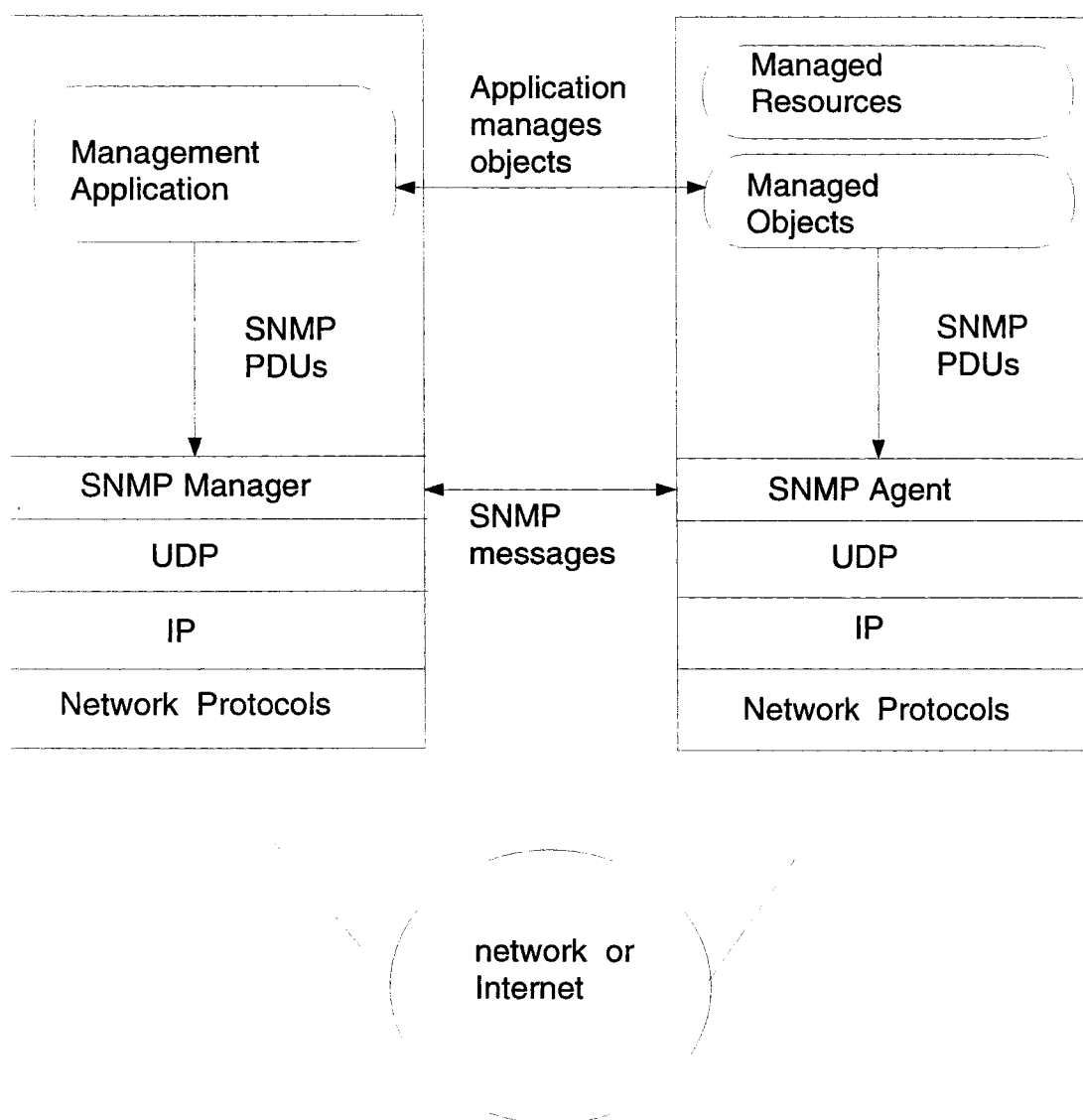


Figure 3.1: Protocol context of SNMP

SetRequest. The first three are variations of the “get” function. All four messages are acknowledged by the agent, who sends a GetResponse message back to the management application. In addition, an agent will issue a Trap message in response to an event that affects the MIB and the underlying managed resources. Because SNMP relies on UDP, which is a connectionless protocol, SNMP is itself connectionless. No ongoing connections are maintained between a management station and its agents. Instead, each exchange is a separate transaction between a management station and an agent.

As with any network management system, the foundation of a TCP/IP based network management system is a database containing information about the elements to be managed. In both the TCP/IP and OSI environments, the database is referred to as a management information base (MIB). Each resource to be managed is represented by a MIB entry, encapsulated as an object. The MIB is a structured collection of such objects. Every agent in the system maintains an MIB that reflects the status of the managed resources at that node. A network management entity monitors the resources at that node by reading the values of objects in the MIB and controls the resources at that node by modifying those values. To support interoperability and improve access a common scheme of representation must be used. The structure of management information provides this representation standard. The structure of management information, SMI, defines the general framework within which an MIB can be defined and constructed. The SMI identifies the data types that can be used in the MIB and how resources within the MIB are represented and named. The philosophy behind SMI is to encourage simplicity and extensibility within the MIB. Thus, the MIB can store only simple data types: scalars and two dimensional arrays of scalars. The SMI does not support the creation or retrieval of complex data structures. This

philosophy is in contrast to that used with OSI management, which provides for the complex data structures and retrieval modes required to support greater functionality. SMI avoids complex data types to simplify the task of implementing basic interoperability.

### 3. MIB structure

Associated with each type of object in an MIB is an identifier of the ASN.1 type OBJECT IDENTIFIER. The identifier serves to name the object. In addition, because the value associated with the type OBJECT IDENTIFIER is hierarchical, the naming convention also serves to identify the structure of object types. An object identifier is a unique identifier for a particular object type. Its value consists of a sequence of integers. The set of defined objects has a tree structure, with the root of the tree being the object referring to the ASN.1 standard. Starting with the root of the object identifier tree, each object identifier component value identifies an arc in the tree. The MIB-II object groups are structured as shown in the Fig 3.2[8].

MIB-II has been proposed as a superset of MIB-I, adding some additional objects and additional groups[8]. The MIB-II objects are subdivided into the following groups:

- . *system* - overall information about the system
- . *interfaces* - information about each of the interfaces from the system to a subnetwork
- . *address translation* - describes address translation table for internet-to-subnet address mapping
- . *ip* - information related to the implementation and execution experience of IP on this system
- . *icmp* - information related to the implementation and execution experience

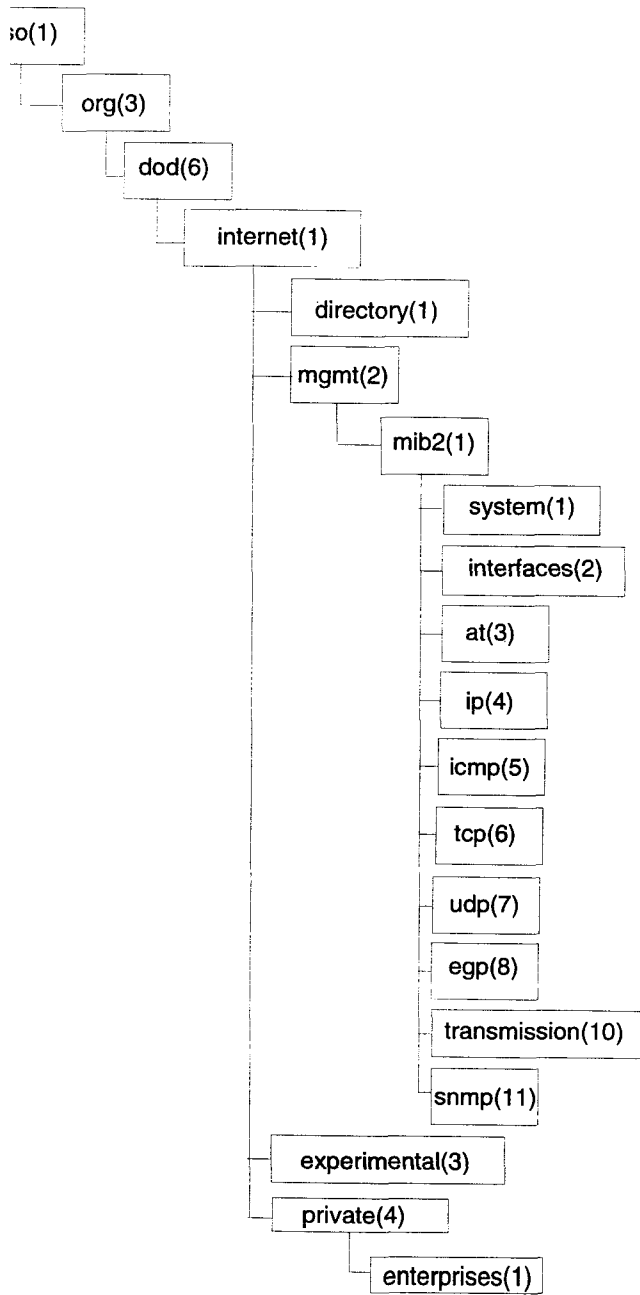


Figure 3.2: MIB2 Object Groups

rience of ICMP on this system

- . *tcp* - information related to the implementation and execution experience of TCP on this system

- . *udp* - information related to the implementation and execution experience of UDP on this system

- . *egp* - information related to the implementation and execution experience of EGP on this system

- . *snmp* - information related to the implementation and execution experience of SNMP on this system

Many users tend to have unrealistic expectations about what can be accomplished with SNMP. A common expectation is that SNMP will enable users to do resource administration and overall system management. But the scope of SNMP is limited by the objects available to manage, namely the MIB or MIB-II. Even MIB-II essentially supports network monitoring and management only from the transport layer down, dealing with how connections get established, how packets are routed, and similar issues. To manage resources such as mail servers and printers, the MIB needs to be further enhanced. SNMP and MIB-II provide a base for gaining experience in doing multivendor network management as well as a limited set of generally useful management features.

## **4. SNMP Formats**

With SNMP, information is exchanged between a management station and an agent in the form of an SNMP message. Each message includes a version number, indicating



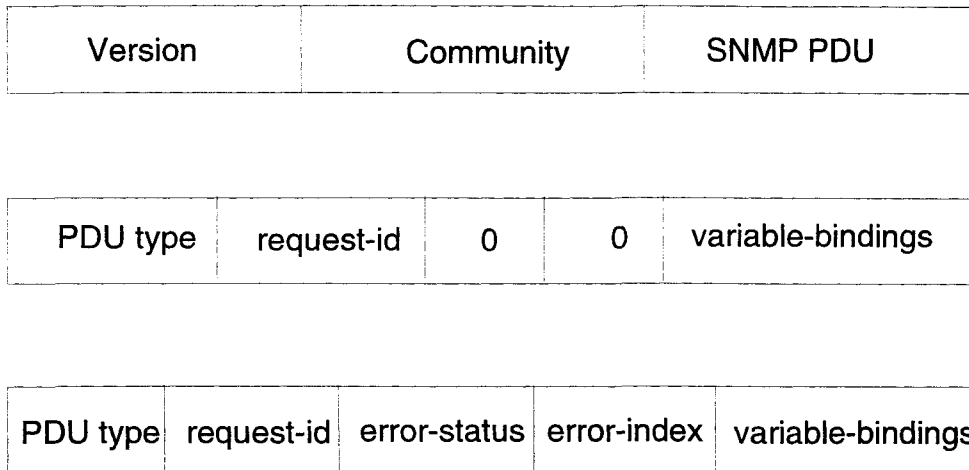


Figure 3.3: SNMP Formats

the version of SNMP, a community name to be used for this exchange, and one of five types of protocol data units (PDUs) as illustrated in Fig 3.3[8].

## 5. Transmission of an SNMP message

As described by Stallings[8], a SNMP entity performs the following actions to transmit PDUs to another SNMP entity.

1. The PDU is constructed, using the ASN.1 structure defined in RFC 1157.
2. This PDU is then passed to an authentication service, together with the source and destination transport addresses and a community name. The authentication service performs any required transformations for this exchange, such as encryption or the inclusion of an authentication code, and returns the result.
3. The protocol entity then constructs a message, consisting of a version

field, the community name, and the result from step 2.

4. This new ASN.1 object is then encoded, using the basic encoding rules, and passed to the transport service.

## **6. Receipt of an SNMP message**

As described by Stallings[8], a SNMP entity performs the following actions upon receipt of a SNMP message.

1. The syntax of the message is checked and the message is discarded if it fails.
2. The version number is verified and if there is a mismatch it is discarded.
3. The protocol entity then passes the user name, the PDU portion of the message, and the source and destination transport addresses to an authentication service. If authentication fails, the authentication service signals the SNMP protocol entity, which generates a trap and discards the message. If authentication succeeds, the authentication service returns a PDU in the form of an ASN.1 object that conforms to the structure defined in RFC 1157.
4. The PDU is checked for the syntax and if it passes the check an appropriate SNMP access policy is selected and the PDU is processed accordingly. The community string in the message acts as a password to authenticate the SNMP message.

## **7. Overview of SNMPv1**

The PDUs supported by version 1 of SNMP are GetRequest PDU, GetNextRequest PDU, SetRequest PDU, Trap PDU and GetResponse PDU. The detailed description

of those are as follows.

The GetRequest PDU is issued by a network management station and in response it gets GetResponse PDU. The GetRequest PDU contains a request-id and variable bindings fields. The variable bindings field contain a list of the object instances whose values are requested. The GetResponse PDU contains the same request-id and it contains the values of the object instances requested. The GetRequest PDU operation is atomic in nature, i.e, either all the values are retrieved or none are. So if the responding entity is able to provide values for all the variables in incoming variable bindings list then the GetResponse PDU includes the variable bindings field with a value supplied to each variable. If at least one of the variable values is not supplied, then no values are returned.

The GetNextRequest PDU is almost identical to the GetRequest PDU. It has the same PDU format and exchange pattern as the GetRequest PDU. The only difference is that in GetRequest PDU the value requested is returned while in GetNextRequest PDU the next value in the lexicographic order is returned. The GetNextRequest PDU is also atomic in nature similar to GetRequest PDU. This PDU provides an efficient mechanism for searching a table whose entries are unknown.

The SetRequest PDU is issued by a network management station. This has the same PDU exchange pattern and format as the GetRequest PDU. The difference between them is that the SetRequest PDU is used to write an object value rather than to read one. The GetResponse PDU contains the request-id and the variable bindings list similar to the other PDUs mentioned above. This is also atomic in nature. Before this PDU is executed the MIB is checked to see whether the object whose value is to

be set has permissions to write. If it does not have appropriate permissions then an error is returned in GetResponse PDU.

The Trap PDU is issued by a network management agent and it is used to provide the management station with an asynchronous notification of some significant event. The format of Trap PDU is quite different from the other PDUs. It contains different fields like *agentaddr*, *generic trap* and *timestamp*. The *agentladdr* is the IP address of the object generating the Trap. The *generic trap* is the type of trap like *ColdStart*, *linkDown*, *linkUp* etc. The *timestamp* is the time at which the trap was generated. Unlike the other PDUs, the Trap PDU does not expect a response from the server.

### **7.1. Limitations of SNMPv1**

There are many limitations for the SNMPv1[8]. SNMP is not suitable for retrieving large volumes of data such as an entire routing table. SNMP traps are unacknowledged. The SNMP MIB model is limited and it does not support applications that are complicated. SNMP does not support manager-to-manager communication, so there is no mechanism that allows a management system to learn about the devices and networks managed by another management system.

## **8. Overview of SNMPv2**

The SNMPv2 was proposed to overcome the deficiencies of SNMPv1[8] . The key enhancements to SNMPv1 are provided in protocol operations, SMI and manager-to-manager communication. The SMI was changed to accomodate complex structures in the database, so complicated applications can be easily supported in the new version.

The most noticeable change in protocol operations is the inclusion of two new PDUs. The `GetBulkRequest` PDU enables the manager to efficiently retrieve large blocks of data. The `InformRequest` PDU enables two managers to talk to each other. Two MIBs are defined as a part of SNMPv2 specification. The SNMPv2 MIB contains basic information, which is analogous to the SNMPv1 MIB-II. The manager-to-manager (M2M) MIB is provided to support M2M communication. Another important change included in SNMPv2 is that the *get* operations are not atomic, i.e, partial results are allowed.

The PDUs supported by SNMPv2 are `GetRequest` PDU, `GetNextRequest` PDU, `GetBulkRequest` PDU, `SetRequest` PDU, `SNMPv2 Trap` PDU and `InformRequest` PDU. The detailed description of these PDUs follows.

### **8.1. GetRequestPDU**

The SNMPv2 `GetRequest` PDU is identical to the SNMP `GetRequest` PDU in format and semantics. The only difference is in the way which responses are handled. The Version 1 operation is atomic in nature, i.e, either all the values are retrieved or none are. If the responding entity is able to provide values for all the variables listed then the response contains the value supplied for each variable. If one of the requested values is not supplied then no values are returned; instead, an error response is supplied. In contrast, in SNMPv2, a variable bindings list is prepared even if values cannot be supplied for all variables. In SNMPv2, a Response PDU is constructed by processing each variable in the incoming variable list, according to the following rules.

1. If the variable does not have an `OBJECT IDENTIFIER` prefix that exactly matches the prefix of any variable accessible by this request, then its value field is set to No-

SuchObject.

2. If the variable's name does not match the name of a variable accessible by this request, then its value field is set to NoSuchInstance.
3. Otherwise, the value field is set to the value of the named variable.

If the processing of a variable name fails then no values are returned. Instead, the responding entity returns a Response PDU with an error status of genErr and a value in the error index field that is the index of the problem object in the variable bindings field.

## **8.2. GetNextRequestPDU**

The SNMPv2 “GetNextRequest” PDU is identical to the SNMP “GetNextRequest” PDU in format and semantics, except for the fact that Version 1 is atomic while Version 2 is not. In SNMPv2, a “Response” PDU for a “GetNextRequest” is constructed by processing each variable in the incoming variable list, according to the following rules.

1. The variable is located that is next in lexicographic order to the named variable. The resulting variable binding pair is set to the name and value of the located variable.
2. If no lexicographic successor exists, then the resulting variable binding pair consists of the name of the variable in the request and a value field set to endOfMibView.

### 8.3. GetBulkRequestPDU

One of the major enhancements provided by SNMPv2 is the “GetBulkRequest” PDU. The purpose of this PDU is to minimize the number of protocol exchanges required to retrieve a large amount of management information. The “GetBulkRequest” PDU allows an SNMPv2 manager to request that the response be as large as possible given the constraints on message size. The “GetBulkRequest” operation uses the same selection principle as the “GetNextRequest” operation; that is, selection is always the next object instance in lexicographic order. The difference is that, with “GetBulkRequest”, multiple lexicographic successors can be selected. In essence, the “GetBulkRequest” operation works as follows. The “GetBulkRequest” includes a list of N variable names in the variable bindings list. For each of the first N names, retrieval is done in the same fashion as for “GetNextRequest”. That is, for each variable in the list the next variable in lexicographic order plus its value is returned. If there is no lexicographic successor then the named variable and a value of `endOfMibView` are returned.

If the processing of a variable name fails for any reason other than `endOfMibView`, then no values are returned. Instead, the responding entity returns a “Response” PDU with an error status of `genErr` and a value in the error index field that is the index of the problem object in the variable bindings field. If the size of the message that encapsulates the generated “Response” PDU exceeds a local limitation or the maximum message size of the request’s source party, then the response is generated with a lesser number of variable binding pairs.

The “GetBulkRequest” operation removes one of the major limitations of SNMP,

which is its inability to efficiently retrieve large blocks of data. Moreover, the use of this operation can actually enable a reduction in the size of management applications that are supported by the management protocol, realizing further efficiencies. There is no need for the management application to concern itself with some of the details of packaging requests. For example, it need not perform a trial-and-error procedure to determine the optimal number of variable bindings to put in a request PDU.

#### **8.4. SetRequest PDU**

The SNMPv2 “SetRequest” PDU is identical to the SNMP “SetRequest” PDU in format and semantics. The only difference is in the way in which responses are handled. The variable bindings are conceptually processed in two phases. In the first phase, each variable binding pair, which constitutes an individual set operation, is validated. If all variable binding pairs are valid, then each variable is altered in the second phase i.e; each individual set operation is performed in the second phase. If no validation errors are encountered, then an attempt is made to update all the variables in the “SetRequest” PDU.

#### **8.5. SNMPv2 Trap PDU**

The SNMPv2 Trap PDU is generated and transmitted by an SNMPv2 entity acting as an agent when an unusual event occurs. This PDU fulfills the same role as the SNMP Trap PDU, but with a different format. The SNMPv2 Trap PDU uses the same format as all other SNMPv2 PDUs except “GetBulkRequest”. As with the SNMP “Trap” PDU, no response is issued to an SNMPv2 “Trap” PDU.



## 8.6. InformRequest PDU

The “InformRequest” PDU is sent by an SNMPv2 entity acting as a manager on behalf of an application to another SNMPv2 entity acting as a manager to provide management information. This PDU is sent to the destinations specified in the SNMPv2 EventNotifyTable which is defined by manager-to-manager MIB. The PDU includes a variable bindings field with the following elements.

1. The first variable is sysUpTime.0, as defined in MIB-II.
2. The second variable is SNMPv2EventID.i, which is defined in the manger-to-manager MIB and which contains the object identifier of the event type.

When an “InformRequest” PDU is received, the receiving SNMPv2 entity first determines the size of a message encapsulating a “Response” PDU with the same values in its request-id, error-status, error-index, and variable bindings fields as the received “InformRequest” PDU. If this size exceeds a local limitation or the maximum message size of the request’s source party, a “Response” PDU is constructed with an error status of tooBig, an error index of 0 and an empty variable bindings field.

If the incoming PDU is not too big, the receiving SNMPv2 entity passes its contents to the destination application and generates a “Response” PDU with the same values in its request-id and variable bindings fields as the received InformRequest PDU, with an error status field of noError and with a value of 0 in its error index field.

## 9. SNMP Installation under Linux

The SNMP which was ported to the Linux host was obtained from a ftp site in Carnegie-Mellon. Version “*cmu-snmp2.1.2.tar.Z*” is a lot different from the previous versions. The changes are briefly described as follows[9].

1. Now *ipNetToMediaType* distinguishes between static and dynamic entries.
2. Man pages are added for all snmp commands.
3. *snmpnetstat* bugs are fixed.
4. Host resources MIB is added.
5. MIB-II snmp group is added.
6. *ipNetToMedia* and *udpEntry* are added to MIB.
7. */etc/snmpd.conf* is added which enables setable location and contact.

The important steps during installing SNMP are: basic setup, compile *source* files, install *config* files and start daemon at boot time.

1. *Basic setup*: First, the *cmu-snmp2.1.2.tar.Z* distribution is uncompressed and then untarred to get the source files. The *asn.1* compiler is installed automatically during this process. Abstract syntax notation one (asn.1) is a formal language developed and standardized by CCITT and ISO. *asn.1* is used to define the structure of application and presentation protocol data units (PDUs). It is also used to define the MIB for SNMP. The *asn.1* compiler parses the *asn.1* notation and converts it into verbose form. For example, something like 1.3.6.1.2.1.6.13 is converted to object *tcpConnTable*.

2. *Compile source files*: The “configure” script file is used to create “Makefile”. The

“configure” script can be used to change the installation path. If it is run without any parameters, then the current directory is taken as the installation path. Then the “Makefile” is run to produce the desirable executables. Needless to say, everything must be done with root privileges. Problems such as missing “include” and “source” files often appear at this point. The installer must determine what files are missing, why they are required, where they should be present, etc. In my case, I had to search the Internet to find those files. I got those files in the patches for this version from a ftp site at the University of North Carolina. As this version is quite new, patches are available separately. Also I have to change some “source” files because of syntax errors. The errors occurred where the source files did not include the “include” files appropriately. Following these changes the compilation worked successfully, producing binary files like *snmpd*, *snmpwalk*, *snmptranslate*, *snmpget*, *snmpset*, *snmpnetstat* and *snmptrapd*.

3. *Install config files:* A default set of *config* files is installed in “/etc” directory by running “installconf” script file. The configuration files installed in “/etc” are *snmpd.conf*, *acl.conf*, *context.conf*, *party.conf* and *view.conf*.

4. *Start up at boot time:* The */etc/rc.d/rc.inet2* file must be changed so that *snmpd*, a daemon that responds to all snmp requests, is started at the boot time. At the boot time, the kernel spawns the process */etc/init*. *init* is a program which reads the configuration file */etc/inittab* and spawns other processes based on the contents of this file. In this process it executes */etc/rc.d/rc.inet2* script which starts daemons like *syslogd*, *inetd*, *snmpd*.

After compiling and configuring everything, other errors can still surface. As an

example, in my installation the error “*needs newer library Version than libc.4.5.2*” occurred when the SNMP binaries were first executed. To remove this error I had to get a new version of the library *libc.4.7.2*, install it, and create symbolic links to point to the new library. When I rebooted the system I found that the symbolic links were still pointing to the old library. Finally, I found information on an Internet newsgroup that helped me to get around this problem. I got a new version of the loader *ld.so.4.7.2*, with new configuration files *ld.so.cache* and *ld.so.config*. I executed *ldconfig* to update the configuration files, then rebooted the system and found that the effective symbolic links now pointed to the new library. This completed the installation of SNMP. The detailed description of the installation and how the executables are used are given in Appendix B. I installed both daemon and client executables on the same machine. It would be interesting to see how the daemon and client executables would work when they are installed on different machines.

## 10. Summary

The SNMPv2 framework is a set of specifications for a second generation SNMP framework. The SNMPv2 SMI provides for more elaborate specification and documentation of managed objects and MIBs. Much of the new material codifies existing SNMP practices. This also includes new macros for defining object groups, traps, compliance characteristics and capability characteristics. This version is available for installation on a range of systems, including the public domain version installed under Linux.

# Chapter 4

## Kerberos

### 1. Introduction

Modern computer systems provide service to multiple users and require the ability to accurately identify the user making a request. In traditional systems, the user's identity is verified by checking a password typed during login; the system records the identity and uses it to determine what operations may be performed. The process of verifying the user's identity is called *authentication*. Password based authentication is not suitable for use on computer networks. Passwords sent across the network can be intercepted and subsequently used by eavesdroppers to impersonate the user[11].

Authentication is the verification of the identity of a party who generated some data, and of the integrity of the data. A *principal* is the party whose identity is verified. The *verifier* is the party who demands assurance of the principal's identity. Data integrity is the assurance that the data received is the same as generated. Authentication mechanisms differ in the number of verifiers: some support a single verifier per message, while others support multiple verifiers.

Other security services include *confidentiality* and *authorization*. Confidentiality is the protection of information from disclosure to those not intended to receive it. Most strong authentication methods optionally provide confidentiality. Authorization is the process by which one determines whether a principal is allowed to perform an operation. Authorization is usually performed after authentication which is based either on information local to the verifier or on authenticated statements by others.

In *authentication by assertion* an application simply asserts the identity of the user to the server, and the server believes it. While more convenient for the user, authentication by assertion hardly qualifies as authentication at all. Such authentication is easily thwarted by modifying the application. Although this requires privileged access to the system, this access is easily obtained on PCs and personal workstations.

Stronger authentication methods base on cryptography are required. Unfortunately, strong authentication technologies are not used as often as they should be, although the situation is gradually improving.

## **2. Kerberos Authentication and Security**

*Kerberos* is a distributed authentication system intended to be used in a large environment. It gives a means for providing service to users from unattended and hence untrusted workstations. It provides integrity and confidentiality for data sent between the client and server. In the Kerberos environment there are three types of hosts. They are client, servers and Kerberos key distribution server. Kerberos was developed at MIT as part of Project Athena and now it is in its fifth version[3]. Kerberos uses DES, a symmetric encryption system. Kerberos was developed before public key

cryptography came into existence. Public key systems have the advantage that it is easier to manage large sets of users using public key cryptography than by using symmetric key systems.

### **3. How Kerberos works**

All the machines in the network using Kerberos rely on it to identify whether the users are “trusted” or not. The primary part of Kerberos is a master database which maintains a list of its users and their private keys. These keys are known only to the Kerberos authentication server. Kerberos uses two types of credentials, called *tickets* and *authenticators*. Both are based on private key encryption technology. A ticket is used to uniquely identify a user to whom it is issued. The ticket also contains information that can be used to verify the user using it. The authenticator contains additional information, which when compared to the information in the ticket proves that the user who has the ticket is the one to whom the ticket was issued[10].

Generally, a ticket is only good for a single service request. Thus each time a new service is requested a new ticket must be obtained. However, in order to reduce network traffic, tickets can be sometimes reused. Timeouts are used to prevent replay by which an attacker steals tickets from the network and then retransmits them at a later time. All the tickets are deleted when the user logs out. If the user stays logged in past a ticket timeout the password must be entered again for authentication. The most obvious problem with Kerberos is that it relies on standard passwords[10]. If the intruder gets access to the password then Kerberos cannot distinguish between the real user and the intruder.

An attacker does not gain any information that would enable him to disguise if authentication based on cryptography was used. Kerberos is the most commonly used example of this type of authentication technology.

## 4. Kerberos Encryption

Though conceptually, Kerberos authentication proves that a client is running on behalf of a particular user, a more precise statement is that the client has knowledge of an encryption key that is known by only the user and the authentication server. The user's encryption key is derived from password in Kerberos. Similarly each application server shares an encryption key with the authentication server.

Encryption in the present implementation of Kerberos uses the data encryption standard (DES). It is a property of DES that if ciphertext (encrypted data) is decrypted with the same key used to encrypt it, the plaintext (original data) appears. If different encryption keys are used for encryption and decryption, or if the ciphertext is modified, the result will be unintelligible, and the checksum in the Kerberos message will not match the data. This combination of encryption and the checksum provides integrity and confidentiality for encrypted Kerberos messages.

## 5. The Kerberos Ticket

Initially the client and server do not share an encryption key. The client relies on the authentication server to generate a new encryption key when it authenticates itself. This new encryption key is called a *session key* and the Kerberos ticket is used to distribute it to the verifier.



The *Kerberos ticket* is a certificate issued by an authentication server and it is encrypted using the server key. The ticket contains a random session key which is used for authentication of the principal to the verifier, the name of the principal to whom the session key was issued and an expiration time after which the session key is not valid. The ticket is not sent directly to the verifier, but is instead sent to the client who forwards it to the verifier as part of the application request. Because the ticket is encrypted in the server key, known only by the authentication server and intended verifier, it is not possible for the client to modify the ticket without detection.

## 6. Application request and response

Kerberos executes in following manner to authenticate the user and to give permission to the user to the end server[3]. First, the user enters the login name in response to the Unix login prompt. Then a message is sent across the network to the Kerberos authentication server before the password prompt. This message contains the login name along with the name of that Kerberos server, the Kerberos ticket granting server (TGS).

$$message = \{login\ name, TGS\ name\}$$

Since this message contains only two names, it is not encrypted. Names are not considered secret since everyone knows those names to communicate. So names are the only entities that are exchanged in cleartext between the workstation and Kerberos. Everything else is encrypted.

The authentication server next checks the login name. If the name is found in the database, the server gets the associated service name. The encryption keys used by Kerberos are used as one-way encrypted passwords, similar to what is stored in

the password field entry of the normal Unix password file. The authentication server forms a response to send back to the *login* program on the workstation. This response contains a ticket that grants the user access to the requested TGS server. Tickets are always sent across the network encrypted. This ticket is a 4-tuple as follows

$$ticket = \{login\ name, TGS\ name, Workstation\ address, TGS\ session\ key\}$$

The workstation address is the Internet address of the workstation. The TGS session key is a random number generated by the authentication server. The authentication server then encrypts this ticket using the encryption key of the TGS server which produces a sealed ticket. A message is then formed consisting of sealed ticket and the TGS session key as follows

$$message = \{TGS\ session\ key, sealed\ ticket\}$$

This message is encrypted using the encrypted password contained in the database.

The *login* program receives the encrypted message and then it prompts for the password. The clear text password entered is encrypted using standard Unix one way encryption function, like *crypt()*, which is the user's encryption key. It is used to decrypt the message that was received. Then the clear text password is erased from the memory. All the workstation have now is the sealed ticket that was encrypted using the TGS encryption key along with the TGS session key. This sealed ticket is incomprehensible to the workstation since it was encrypted using the TGS encryption key which is known only to the TGS and Kerberos. At this point the workstation saves a copy of the sealed ticket and the TGS session key. Every workstation contains similar information, a sealed ticket for the ticket granting service along with the corresponding random TGS session key. Even though all the sealed tickets are encrypted using the same encryption key, since each ticket contains the name of the user the ticket was granted to, the workstation address, a different random TGS session key,

each one is different.

The workstation builds a message to be sent to the ticket granting service. This message is a 3-tuple as follows

$$message = \{sealed\ ticket, sealed\ authenticator, end\ server\ name\}$$

The authenticator is created by the workstation and it is a 3-tuple as follows

$$authenticator = \{login\ name, Workstation\ address, current\ time\}$$

The authenticator is sealed using the TGS session key obtained from the TGS server. The ticket granting service receives the message and then it decrypts the sealed ticket using the TGS encryption key. From the unencrypted ticket the TGS gets the TGS session key. It uses this session key to decrypt the sealed authenticator. There are lots of items for the TGS to check for validity. They are the login name in both the ticket and the authenticator and the TGS server name in the ticket. Also it checks the network address in the ticket and the authenticator. Finally, it compares the current time in the authenticator to check whether the message was in the valid time range. This requires that all the workstations and servers maintain their clocks within some prescribed tolerance. The TGS now gets the end server name from the message in the Kerberos database. It also gets the encryption key for the specified service. Then the TGS forms a new random session key and then creates a new ticket based on the requested end service name and the new session key as follows

$$ticket = \{login\ name, end\ server\ name, Workstation\ name\ new\ session\ key\}$$

This ticket is sealed using the encryption key for the requested end server. The TGS forms a message as follows

$$message = \{new\ session\ key, sealed\ ticket\}$$

This message is then sealed using the TGS session key and sent to the client workstation. The client decrypts this using the TGS session key. The client builds an

authenticator as follows

$$authenticator = \{login\ name, Workstation\ name, current\ time\}$$

It seals this using the new session key. Then it builds a message as follows

$$message = \{sealed\ ticket, sealed\ authenticator, end\ server\ name\}$$

This message is not encrypted as both the ticket and the authenticator are already sealed and the other one is just the end server name. The end server receives this message and decrypts the sealed ticket using its encryption key. It then uses the new session key contained in the ticket to decrypt the authenticator. Then it does some validation like checking the login name, the workstation address and the current time. Then if everything is valid, it communicates with the client.

## 7. Obtaining additional tickets

The basic Kerberos authentication protocol allows a client with knowledge of the user's password to obtain a ticket and session key for and to prove its identity to any verifier registered with the authentication server. The user's password must be presented each time the user performs authentication with a new verifier. This is cumbersome and hence instead the system should support single sign-on where the user logs into the system once and subsequent authentication occurs automatically. The obvious way to support this is to cache the user's password on the workstation. However this is dangerous. Though a Kerberos ticket and the key associated with it are valid for only a short time, the user's password can be used to obtain tickets, and to impersonate the user until the password is changed. Hence, Kerberos caches only tickets and encryption keys, collectively called as *credentials* which work only for a limited period.

The ticket granting exchange of the Kerberos protocol allows a user to obtain tickets and encryption keys using such short-lived credentials, without re-entry of the user's password. When the user first logs in, an authentication request is issued and a ticket and session key for the ticket granting service is returned by the authentication server. This ticket, called a ticket granting ticket, has a relatively short life. The response is decrypted, the ticket and session key are saved, and the user's password forgotten.

Subsequently, when the user wishes to prove its identity to a new verifier, a new ticket is requested from the authentication server using the ticket granting exchange. The ticket granting exchange is identical to the authentication exchange except that the ticket granting request has embedded within it an application request, authenticating the client to the authentication server, and the ticket granting response is encrypted using the session key from the ticket granting ticket, rather than the user's password.

## **8. Protecting application data**

As described so far, Kerberos provides only authentication: assurance that the authenticated principal is an active participant in an exchange. A by-product of the Kerberos authentication protocol is the exchange of the session key between the client and the server. The session key may subsequently be used by the application to protect the integrity and privacy of communications. The Kerberos system defines two message types, the safe message and the private message to encapsulate data that must be protected, but the application is free to use a method better suited to the particular data that is transmitted.

## 9. Infrastructure and Cross-Realm Authentication

In a system that crosses organizational boundaries, it is not appropriate for all users to be registered with a single authentication server. Instead, multiple authentication servers must exist, where each is responsible for a subset of the users or servers in the system. The subset of the users and servers registered with a particular authentication server is called a *realm* (if a realm is replicated, users will be registered with more than one authentication server). Cross-realm authentication allows a principal to prove its identity to a server registered in a different realm.

To prove its identity to a server in a remote realm, a Kerberos principal obtains from its local authentication server a ticket granting ticket for the remote realm. This requires the principal's local authentication server to share a cross-realm key with the verifier's authentication server. The principal next uses the ticket granting exchange to request a ticket for the verifier from the verifier's authentication server, which detects that the ticket granting ticket was issued in a foreign realm, looks up the cross-realm key, verifies the validity of ticket granting ticket, and issues a ticket and session key to the client. The name of the client, embedded in the ticket, includes the name of the realm in which the client was registered.

With Kerberos Version 4, it was necessary for an authentication server to register with every other realm with which cross-realm authentication was required. This was not scalable because complete interconnection required the exchange of  $n^2$  keys where  $n$  was the number of realms.

In contrast, Version 5 supports multi-hop cross-realm authentication, allowing keys to

be shared hierarchically. With V5, each realm shares a key with its children and parent. For example, the UMT.EDU realm shares a key with the EDU realm which also shares keys with MIT.EDU, USC.EDU and UMN.EDU. If no key is shared directly by UMT.EDU and MIT.EDU, authentication of the client `vamc@UMT.EDU` to a server registered with the MIT.EDU realm proceeds by obtaining a ticket granting ticket for EDU from the UMT.EDU authentication server, using that ticket granting ticket to obtain a ticket granting ticket for the MIT.EDU realm from the EDU authentication server and finally obtaining a ticket for the verifier from the MIT.EDU authentication server.

The list of realms that are transmitted during multi-hop cross-realm authentication is recorded in the ticket and the verifier accepting the authentication makes the final determination about whether the path that was followed should be trusted. Shortcuts through the hierarchy are supported and can improve both the trust in and the performance of the authentication process.

## **10. Other approaches for improving security**

Kerberos is not a complete solution to network security problems. There are some other tools which provide partial solutions to the network security problems and when combined with Kerberos they provide stronger security. Among these other tools are one-time passcodes and public-key cryptography[12].

## **11. One-time passcodes**

A one-time passcode authentication mechanism uses a different passcode each time authentication is required. Kerberos does not protect against the theft of a password

through a Trojan horse login program on the user's workstation, but if the user's password were to change each time it was entered, a password stolen in this manner would be useless to an attacker. One-time passcode authentication methods typically use a credit card sized device that either displays a time varying password (called a passcode), or returns a passcode when a challenge is entered on a small keypad. Some methods use a printed list of passcodes that can be used one after another. When a user logs in using one of these devices, the user is prompted for the passcode. Depending on the style, the prompt may include the challenge that is to be typed into the device. The user enters the passcode from the device in much the same way as a normal password.

One-time passcode methods can be combined with Kerberos so that knowledge of both the passcode and a password-based encryption key are required to successfully complete the initial authentication exchange. Commercial products that combine one-time passcodes with Kerberos are available.

## **12. Public-key Cryptography**

In public-key cryptography, encryption and decryption are performed using a pair of keys such that knowledge of one key does not provide knowledge of the other key in the pair. One key is published and is called the public key, and the other key is kept private. This second key is called the private key, not to be confused with a secret key which is shared by the parties to communication in a conventional cryptosystem (it takes two to share a secret, but once you tell something to anyone else it is no longer private). Public-key cryptography has several advantages over conventional cryptography when used for authentication[14]. These include more natural support



for authentication to multiple recipients and the elimination of secret encryption keys from the central authentication server.

While public-key encryption is well suited for use in authentication by store and forward applications such as electronic mail, and it is required by applications where a signature is verified by many readers, performance is a problem for high-performance servers that perform many authentication operations. With the RSA algorithm, the most accepted algorithm for public key cryptography, the private key operation (signing or decrypting a message) is expensive.

Lots of work has also been done to add public key support to Kerberos where it can be confined to the initial request for a ticket granting ticket. Subsequent exchanges use normal cryptography for better performance. Public key encryption can also be used by authentication servers to exchange conventional cross realm keys.

### **13. Limitations of Kerberos**

There are several problems associated with Kerberos[15]. One of the most serious problems is that the Kerberos master authentication server must store all the keys in plain text. So if the security of the master authentication server is compromised then the entire Kerberos system is compromised. Also the Kerberos is not secure against replay attacks. This is because authenticators have time outs associated with them which are large enough to accommodate machines with slightly unsynchronized clocks. This could be solved by maintaining a cache of all *used* authenticators so that no time outs are necessary.

In particular, Kerberos is not effective against password guessing attacks; if a user chooses a poor password, then an attacker guessing that password can impersonate the user. Similarly, Kerberos requires a trusted path through which passwords are entered. If the user enters a password to a program that has already been modified by an attacker (a Trojan horse), or if the path between the user and the initial authentication program can be monitored, then an attacker may obtain sufficient information to impersonate the user

## 14. Kerberos Installation under Linux

The version of Kerberos ported to Linux was obtained from a ftp site at MIT. The important steps during installation of Kerberos are: set up, configuration, build, and installation[13].

1. *Set up:* In order to build and install Kerberos approximately 60-70 MB of disk space is needed. The exact amount varies depending on the platform and whether the distribution is compiled with debugging options or not. The first step is to guarantee that the required space is available to unpack the source distribution. The Kerberos V5 distribution comes in two compressed tar files. The first file, which is generally named *krb5.src.tar.gz*, contains the sources for all of Kerberos except for the crypto library, which is found in the file *krb5.crypto.tar.gz*.

2. *Configuration:* There are a number of options to “configure” to control how the Kerberos distribution is built. Options like compiler, flags and root directory are configured before compiling the source files. Here the “gcc” compiler is used and all the default set of “gcc” compiler flags are used. The files to be configured are “os-

conf.h”, “config.h”, “krb.conf” and “krb.realms”. “osconf.h” and “config.h” are two configuration files which should be edited to control various compile time parameters in the Kerberos distribution. Several things like pathname to the file which defines the known realms, pathname to the file which assigns hosts to realms and pathname to the database that maps authentication names to local account names are defined in “osconf.h”. Several error messages are defined in “config.h”. These “configure” files are generated using “autoconf” which is found in the “/src/util/autoconf” directory in the distribution. The “krb.conf” file is used to specify a system’s default Kerberos realm and locations of the Kerberos servers. Here the realm name is “UMT.EDU”. All the realms and the mapped hosts are mentioned in “krb.realms” file. All hosts which uses Kerberos should have certain ports defined in the system’s “/etc/services” file.

3. *Build*: The build is performed using standard “make” tools. After configuring all the files, the supplied “Makefile” is executed which should produce all the executables required for the Kerberos system. There are lots of syntax errors in the source code and they are fixed by me during compiling process.

4. *Installation*: The realm name must be chosen to specify the system’s administrative domain. Usually the realm name is the Internet domain name in capital letters. Kerberos realm names are case-sensitive. For example, if the Internet domain name is *umt.edu*, then the Kerberos realm name should be “UMT.EDU”. In my installation, I discovered several specific problems after first installing the software. When I tried to run the test program it did not work in the way it is supposed to. These problems had also been reported when Kerberos was ported to Ultrix 4.3 machines[20], and were traced to the file “src/lib/crypto/descrc.c”. Internet sources[20] provided infor-

mation on how to fix these problems. Also I had problems because the realm name was not recognized by the Kerberos database. This is also fixed with help from the Internet sources[20]. After all these fixes, Kerberos works fine on the Linux machine.

## **15. Summary**

The Version 5 Kerberos is the latest available version of a “standard” public domain authentication service used to verify users. This version is available for installation on a range of systems. Kerberos can be used with SNMP to provide more security to the SNMP system.

# Chapter 5

## Firewalls

### 1. Introduction

A *firewall* is a term derived from a part of a car. In cars, firewalls are physical objects that separate the engine block from the passenger compartment. They are meant to protect the passenger in case the car explodes. A firewall in computers is a logical device that limits communication between two networks[7].

The simplest form of firewall would be the following.

1. Take a computer that has routing capabilities, such as a system running Unix.
2. Put in two interfaces, eg., serial ports, ethernet, token ring, etc.
3. Connect one network to each interface.
4. Impose a selective form of network layer forwarding to link traffic between the two networks.

The purpose of a firewall is to provide a point of defense which controls access to services. This requires a mechanism for selectively permitting or blocking traffic between two networks. Most firewalls use a combination of *proxying* and *packet filtering*.

Proxying is done using “proxy servers”. A proxy for a network protocol is an application that runs on a firewall host and connects specific service requests across the firewall acting as a gateway. The proxy protects the firewall host by eliminating the need for the user to log into the firewall. Packet filtering is a process through which the firewall host selectively allows packets to pass through the firewalls. All the network traffic is analyzed in the form of network layer packets in the filtering gateway. Each packet is examined to match a set of filtering rules and dealt accordingly. Generally, packets are allowed to pass freely from the internal network to the outside world, but packets from the outside world are filtered out of the traffic visible to the internal network. There are two types of firewalls. They are internal firewalls and external firewalls. An external firewall is placed in between the external network and the internal network while the internal firewall is placed in between internal network of non-sensitive systems and internal network of sensitive systems.

## **2. What can a firewall protect against ?**

Some firewalls permit only Email traffic to pass between networks, thereby protecting the protected network against any attacks other than attacks against the Email service. Other firewalls provide less strict protections, and block services that are known to be subject to security problems.

Generally, firewalls are configured to prevent any access of internal services from the “outside” world. This, more than anything, helps prevent rouges from logging into machines on a protected network from the outside. More elaborate firewalls block traffic from the outside to the inside, but permit users on the inside to communicate freely with the outside.

Firewalls are also important since they can provide a single “choke point” where security and audit can be imposed. Unlike in a situation where a computer system is being attacked by someone dialing in with a modem, the firewall can act as an effective “phone tap” and tracing tool.

### **3. What can’t a firewall protect against ?**

Firewalls can’t protect against attacks that don’t go through the firewall. Many corporations that connect to the Internet are very concerned about proprietary data leaking out of the company through that route. Unfortunately for those concerned, a magnetic tape can just as effectively be used to export data. Firewall policies must be realistic, and reflect the level of security in the entire network. For example, a site with top secret or classified data doesn’t need a firewall at all: they shouldn’t be hooking up to the Internet in the first place, or the systems with the really secret data should be isolated from the rest of the corporate network.

Firewalls can’t protect very well against things like viruses[17]. There are too many ways of encoding binary files for transfer over networks, and too many different architectures and viruses to try to search for them all. In other words, a firewall cannot replace security-consciousness on the part of a network’s users. In general, a firewall cannot protect against a data-driven attack – attacks in which something is mailed or copied to an internal host where it is then executed. For example, this form of attack has occurred in the past against various versions of Sendmail, and once a user on the protected network has accepted such a mail message, the attack has by-passed the firewall security.

## 4. Design Decisions

In configuring a firewall, the major design decisions with respect to security are often already dictated by corporate or organizational policy. Specifically, a decision must be made as to whether security is more important than ease-of-use, or vice versa. There are two basic approaches that summarize the conflict:

1. That which is not expressly permitted is prohibited
2. That which is not expressly prohibited is permitted

The importance of this distinction cannot be overemphasized. In the former case, the firewall must be designed to block everything, and services must be enabled on a case-by-case basis only after a careful assessment of need and risk. This tends to impact users directly, and they may see the firewall as a hindrance. In the second case, the systems administrator is placed in a reactive mode, having to predict what kinds of actions the user population might take that would weaken the security of the firewall, and having to prepare defenses against them. This essentially pits the firewall administrator against the users in an endless arms race that can become quite fierce. Users can generally compromise the security of their login if they try or aren't aware of reasonable security precautions. If the user has an open access login on the firewall system itself, a serious security breach can result. The presence of user logins on the firewall system tends to magnify the problem of maintaining the system's integrity. A second important statement of policy is implicit in the *that which is not expressly permitted is prohibited* stance. This stance is more "fail safe", since it accepts that the administrator is ignorant of what TCP ports are safe, or what holes may exist in the manufacturer's kernel or applications. Since many vendors are slow to publicise



security holes, this is clearly a more conservative approach. It is an admission of the fact that what you don't know CAN hurt you.

## 5. About Proxy Servers

Firewalls inhibit the access to the Internet from the private network. Programs like Netscape, which require a direct Internet connection, will not work in normal mode from behind the firewall. Being unable to ftp directly to your computer is another big problem, requiring a two step, Internet to firewall and firewall to protected computer setup. These problems can be overcome by using a *Proxy Server*.

In a proxy server the data is handled at an application level rather than at the packet level. A proxy server looks and acts like a real server from the client's point of view and as a real client from the real server's point of view. For example, programs like Netscape can be configured to use a proxy server so that they also allow the addition of strong user authentication which is not possible in packet filtering gateways. One of the new mechanisms provided by proxy servers is "caching". The caching mechanism is disk based and persistent which means that it survives restarts of the proxy server as well as the server itself. The proxy caching provides "virtual bandwidth".

## 6. Internal Components of a Firewall

The important internal components of a firewall are the *screening router* and *bastion host*[18]. A screening router is a basic component of most firewalls. A screening router can be a commercial router or a host-based router with some kind of packet filtering capability. Typical screening routers have the ability to block traffic between

networks or specific hosts, on an IP port level. Some firewalls consist of nothing more than a screening router between a private network and the Internet. The term bastion originally referred to the highly fortified parts of a medieval castle, i.e., a point that overlooks critical areas of defense and usually has stronger walls, room for extra troops, and the occasional useful tub of boiling hot oil for discouraging attackers. A bastion host is a system identified by the firewall administrator as a critical strong point in the network's security. Generally, bastion hosts will have some degree of extra attention paid to their security, will undergo regular audits, may have modified software, and may have restrictions on the way it is used. The bastion hosts should also be physically secure.

## **7. Typical Screened Host Gateway**

Possibly the most common firewall configuration is a screened host gateway. This is implemented using a screening router and a bastion host as illustrated in Figure 5.1[18]. Usually, the bastion host is on the private network, and the screening router is configured such that the bastion host is the only system on the private network that is reachable from the Internet. The bastion host thus needs to maintain a high level of host security. Often the screening router is configured to block traffic to the bastion host on specific ports, permitting only a small number of services to communicate with it. There are some disadvantages to this architecture. The major one is that if the attacker manages to break in to the bastion host, then there is nothing left in the way of network security between the bastion host and the rest of the internal hosts[21].

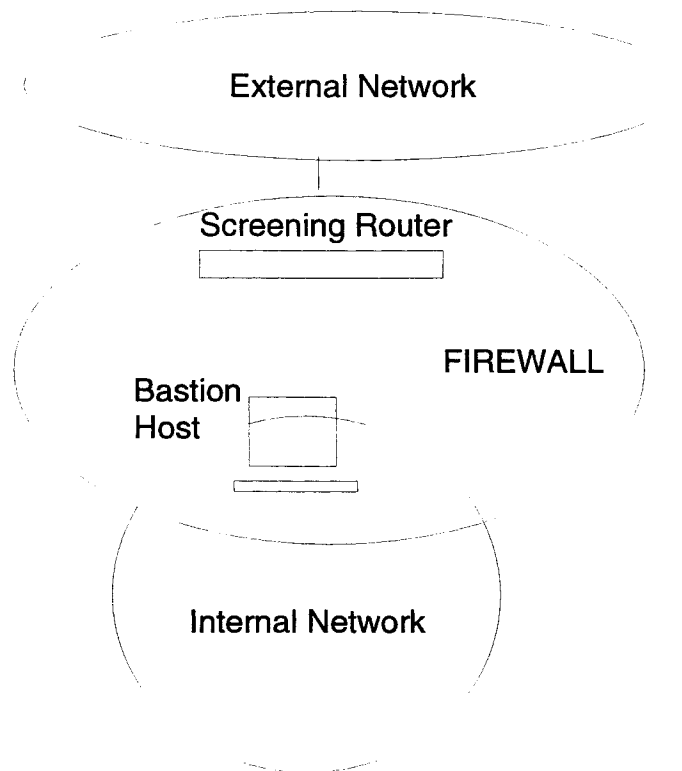


Figure 5.1: Typical Screened Host Gateway

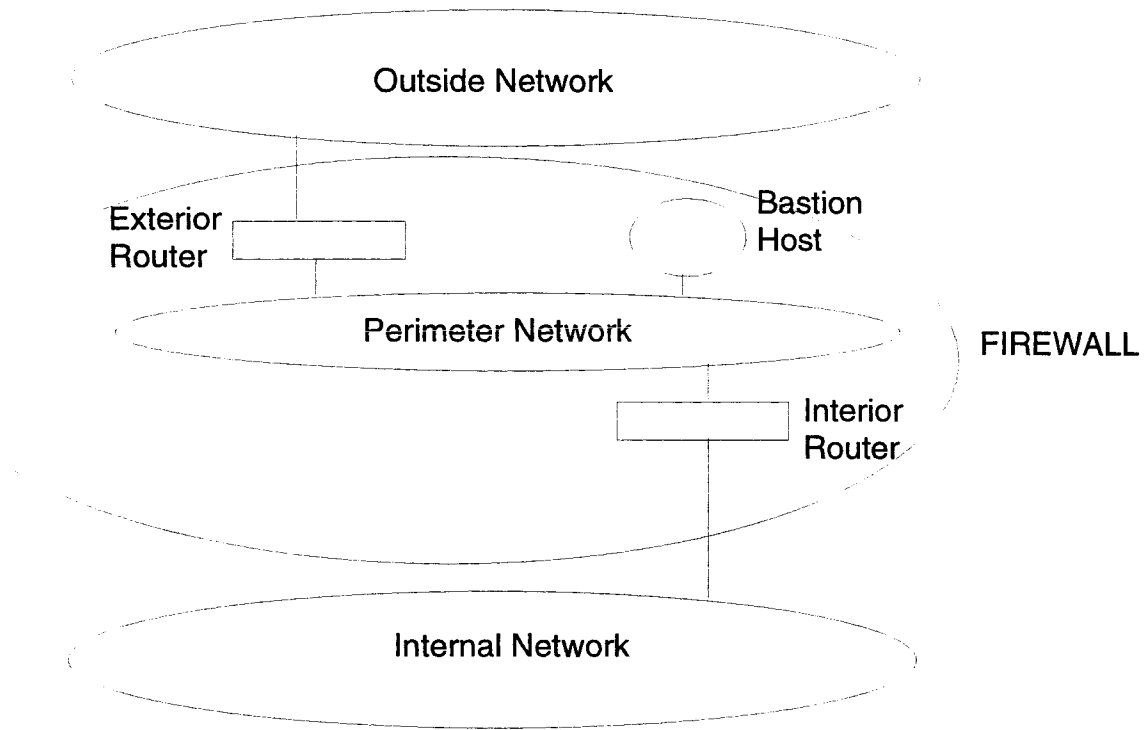


Figure 5.2: Typical Screened Subnet

## 8. Typical Screened Subnet

In some firewall configurations, an isolated subnet is created, situated between the Internet and the private network. This adds an extra layer of security to the screened host architecture. Typically, this network is isolated using screening routers, which may implement varying levels of filtering. Generally, a screened subnet is configured such that both the Internet and the private network have access to hosts on the screened subnet, but traffic across the screened subnet is blocked. Some configurations

of screened subnets will have a bastion host on the screened network, either to support interactive terminal sessions or application level gateways as illustrated in Fig 5.2[18].

The two main culprits attacking firewalls are *IP spoofing* and *hijacking tools*, and combinations of both. Generally, intruders use IP spoofing to gain root access for some purpose. Similarly, they can hijack terminal connections regardless of their method of gaining root access.

## 9. IP spoofing

To gain access, intruders create packets with spoofed source IP addresses, attempting to exploit applications that use authentication based on IP addresses. IP spoofing thus leads to unauthorized user and possibly root access on the targeted system. It is possible to route packets through filtering-router firewalls if they are not configured to filter incoming packets whose source address is in the local domain. It is important to note that IP spoofing attacks are possible even if no reply packets can reach the attacker.

Examples of configurations that are potentially vulnerable include

1. routers to external networks that support multiple internal interfaces
2. routers with two interfaces that support subnetting on the internal network
3. proxy firewalls where the proxy applications use the source IP address for authentication

Familiar services that are vulnerable to the IP spoofing attack include SunRPC and NFS, X windows exchange protocols, and all other applications that use source IP

addresses for authentication.

## 10. Hijacking tools

Once the intruders have root access on a system, they can use a tool to dynamically modify the UNIX kernel. This modification allows them to hijack existing terminal and login connections from any user on the system[20].

In taking over the existing connections, intruders can bypass one-time passwords and other strong authentication schemes by tapping the connection after the authentication is complete. For example, a legitimate user connects to a remote site through a login or terminal session. The intruder hijacks the connection after the user has completed the authentication to the remote location, and the remote site is now compromised.

Currently, hijacking tools are most often used on SunOS 4.1.x systems due to the wide number of these systems available and the fact that there are well known security leaks on these systems that allow unauthorized root access. However, the system features that make this type of attack possible are not unique to SunOS.

Intruder activity based on IP spoofing source IP addresses can lead to unauthorized remote root access even to systems behind a filtering-router firewall. After gaining root access and taking over existing terminal and login connections, intruders can also gain access to remote hosts.

## 11. Solutions

The first step in “solving” or avoiding IP spoofing problems is simply to detect these attacks when they occur[20]. If packets are monitored using network monitoring software such as *netlog* and if a packet on the external interface has both its source and destination IP addresses in the local domain then the system is under attack. Another way to detect IP spoofing is to compare the process accounting logs between systems on the internal network. If the IP spoofing attack has succeeded, a log entry on the victim machine showing a remote access is found on the apparent source machine and there will be no corresponding entry for initiating that remote access.

Detecting the use of hijacking tools involves the following steps. When the intruder attaches to an existing terminal or login connection, users may detect unusual activity, such as commands appearing on their terminal that they did not type or a blank window that will no longer respond to their commands. Users should be encouraged to inform any such activity to system administrator. In addition particular attention should be paid to connections that have been idle for a long time. Once the attack is completed, it is difficult to detect. However, the intruders may leave remnants of their tools. For example, a kernel streams module designed to tap into existing TCP connections can be found.

The next step is the implementation of preventive measures designed to reduce the chance that attacks of all types will be succesful. The best method of preventing the IP spoofing problem is to install a filtering router that restricts the input to an external interface, known as an *input filter*. This technique does not allow a packet through if it has a source address from internal network. In addition, outgoing pack-

ets that have a source address different from the internal network are filtered in order to prevent a source IP spoofing attack originating from the site. Several vendors have reported support for this feature; including Bay Networks/Wellfleet routers (Version 5 and later), Cabletron - LAN Secure, Cisco - RIS software (all releases of Version 9.21 and later), and Livingston (all Versions)[20].

If a vendor's router does not support filtering on the inbound side of the interface or if there is a delay in incorporating the feature in the system, the spoofed IP packets can be filtered by using a second router between external interface and the outside connection. This second router is configured to block all the packets that have a source address in the internal network. To serve this purpose a filtering router or a Unix system with two interfaces that supports packet filtering can be used.

There is no specific way to prevent use of hijacking tools other than preventing intruders from gaining root access in the first place. If there is a root compromise that permits a hijacking recovery following the attack can be done as follows.

1. Disconnect from the network or operate the system in single-user mode during the recovery. This will keep users and intruders from accessing the system.
2. Verify system binaries and configuration files against the vendor's media. Do not rely on timestamp information to provide an indication of modification. Do not trust any verification tool such as `cmp(1)` located on the compromised system as it too may have been modified by the intruder. In addition, do not trust the results of the standard UNIX `sum(1)` program, as intruders have been known to modify system files in such a way that the checksums remain the same. Replace any modified files from the vendor's media, not from backups.
3. Search the system for new or modified `setuid` root files.



```
find / -user root -perm -4000 -print
```

If the type of filesystem is NFS or AFS, use `ncheck` to search the local file systems.

```
ncheck -s /dev/sd0a
```

4. Change the password on all accounts.
5. Don't trust backups for reloading any file used by root.

## **12. Summary**

In this chapter a brief description of what the firewalls are and what they can and cannot do are provided. Also IP spoofing and ways to overcome it are described. Presently, there are no standard public domain firewall systems that can easily be installed and tested in a Linux environment. As a result, I did not implement any firewall installation. Proxy servers are discussed which allow direct access to the Internet from behind a firewall. Firewalls can be used with SNMP or Kerberos to provide additional security to the system.

# Bibliography

- [1] Kirch, Olaf. *Linux Network Administrator's Guide*, O'Reilly and associates Inc.
- [2] Curry, David. *UNIX system security-A guide for users and system administrators*, Addison Wesley Professional computing series.
- [3] Stevens, Richard. *Unix network programming*, Prentice Hall.
- [4] *Linux Networking Manual*, Slackware Inc., California.
- [5] Comer, Douglas. *Internetworking with TCP/IP, volume 1*, Prentice Hall.
- [6] Comer, Douglas, David Stevens. *Internetworking with TCP/IP, volume 2*, Prentice Hall.
- [7] Hunt, Craig. *TCP/IP networking*. O'Reilly and Associates.
- [8] Stallings, William. *SNMP, SNMPv2 and CMIP, The practical guide to network management standards*, Addison Wesley computing series.
- [9] *SNMP notes from Carnegie Mellon*.
- [10] Needham and Shroeder. *Kerberos Authentication system*.
- [11] Neuman, Clifford and Ts'o, Theodore. *An Authentication Service for Computer Networks*, IEEE Communications, 32(9):33-38. September 1994.
- [12] Zorn, Glen and Neuman, Clifford. *One-time passwords with Kerberos*, Internet Draft ietf-cat-kerberos-passwords-01, April 1995.
- [13] *Kerberos notes from MIT*.
- [14] Wray, Neuman and Tung. *Public key Cryptography for Initial Authentication in Kerberos*, Internet Draft ietf-cat-kerberos-pk-init-00, March 1995.
- [15] Bellovin and Merritt. *Limitations of the Kerberos Authentication System*. Proceedings of the Winter 1991 Usenix Conference. January 1991.
- [16] Garfinkel, Simson and Spafford, Gene. *Building Internet Firewalls*, O'Reilly and Associates, June, 1991.
- [17] Cheswick, William and Bellovin Stephen. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, Spring 1994.

- [18] Ranum, Marcus. *Thinking About Firewalls*, Proceedings of Second International Conference on Systems and Network Security and Management(SANS), April, 1993.
- [19] Ranum, Marcus. *An Internet Firewall*, Proceedings of First International Conference on Systems and Network Security and Management(SANS), November, 1992.
- [20] *Sources on Internet*.
- [21] Chapman, Brent and Zwicky, Elizabeth. *Building Internet Firewalls*, O'Reilly and Associates.

## Appendix A

### Linux Installation

The hard drive is partitioned using the *fdisk* program as follows, under Linux

*fdisk /dev/sda*

where */dev/sda* is the drive name. This gives a menu as follows.

*Command (m for help): m*

*Command action*

*a toggle a bootable flag*

*d delete a partition*

*l list known partition types*

*m print this menu*

*n add a new partition*

*p print the partition table*

*q quit without saving changes*

*t change a partition's system id*

*u change display/entry units*

*v verify the partition table*

*w write table to disk and exit*

*x extra functionality*

Here *n* is selected to create a new partition. This gives the following menu.

*Command (m for help): n*

*Command action*

*e extended*

*p* *primary partition (1-4)*

Primary partition is created by selecting *p*.

The partition number 2 is entered, because the first partition is already used by DOS.

The amount of space required is mentioned by me.

The *swap* partition is created similarly.

Now the swap partition is activated as follows.

```
mkswap -c /dev/sda3 10336
```

After formatting the swap space it is enabled as follows.

```
swapon /dev/sda3
```

The filesystem is created as follows.

```
mke2fs -c partitionname size
```

where *partitionname* is the name of the partition and *size* is the size of the partition in blocks. Now the system is ready for the installation of Linux. The “*setup*” program is used to install Linux where the source and specific software to be installed are selected.

The network is configured using the *netconfig* command. The IP address and netmask are defined here which are added to the */etc/hosts* file. The system is loaded by the Linux loader (*lilo*). The system has to be configured to tell the loader which operating system to load while booting. After configuring the system the file */etc/lilo.conf* is produced. In order to activate the system */etc/lilo* is executed. The system can be rebooted by using either *reboot* or *shutdown -r now*.

## Appendix B

### SNMP Installation

The SNMP ported to Linux host was obtained from a ftp site in Carnegie Mellon. First, the *cmu-snmp2.1.2.tar.gz* is uncompressed and untarred to get the source files as follows.

```
gzip -d cmu-snmp2.1.2.tar.gz
tar -xvf cmu-snmp2.1.2.tar
```

The *asn.1* compiler is installed automatically during this process.

The *configure* script file is used to create *Makefile*. To compile *make* is executed. Only client or server part (agent) can be installed by executing *make installclient* or *make installagent*. The configuration files are installed in */etc* directory by running *installconf* as follows.

```
/etc/installconf hostname ipaddress
```

The configuration files installed in */etc* are *snmpd.conf*, *acl.conf*, *context.conf*, *party.conf* and *view.conf*.

The executables produced after compiling are *snmpwalk*, *snmptranslate*, *snmptrapd*, *snmpget*, *snmpnetstat*, *snmpd* and *snmpset*. All these binaries are kept in */usr/sbin* directory. The */etc/rc.d/rc.inet2* file must be changed so that *snmpd*, a daemon that responds to all snmp requests, is started at the boot time as follows.

```
if [ -f /usr/sbin/snmpd ]
then
echo -n "snmpd"
```

```

/usr/sbin/snmpd &
else
echo "snmpd not found"
exit 1
fi

```

The SNMP executables are used as described below. *snmpget* is used as follows.

```
snmpget [-v 2] hostname noAuth objectID [objectID]...
```

For example,

```
snmpget darkstar.umd.edu noAuth system.sysName.0
```

retrieves the variable *system.sysName.0*.

*snmpnetstat* is used as follows.

```
snmpnetstat hostname noAuth -[airp]
```

For example,

```
snmpnetstat darkstar.umd.edu noAuth -a
```

displays the state of all sockets.

```
snmpnetstat darkstar.umd.edu noAuth -i
```

displays the state of all interfaces.

```
snmpnetstat darkstar.umd.edu noAuth -r
```

displays the routing tables.

```
snmpnetstat darkstar.umd.edu noAuth -p
```

displays the statistics about a particular protocol.

*snmpset* is used as follows.

```
snmpset darkstar.umd.edu noAuth system.sysContact.0
```

sets the variable *system.sysContact.0*.

*snmptranslate* is used as follows.

```
snmptranslate -[n d]
```

For example,

```
snmptranslate -n sysdescr
```

translates *sysdescr* to actual form like *system.sysDescr*.

```
snmptranslate -d sysdescr
```

prints description of the object.

*snmpwalk* is used as follows.

```
snmpwalk darkstar.umat.edu noAuth system
```

retrieves all the variables under *system* in the MIB. This is like *SNMP GetBulk request* of SNMPv2.



## Appendix C

### Kerberos Installation

The Kerberos V5 distribution consists of two files *krb5.src.tar.gz* and *krb5.crypto.tar.gz*. Both files are uncompressed and untarred as follows.

```
gzip -d krb5.src.tar.gz
```

```
tar -xvf krb5.src.tar
```

There are a number of options to *configure* to control how the Kerberos distribution is built. Options like compiler, flags and root directory are configured before compiling the source files. The files that are configured are *osconf.h*, *config.h* and *krb.conf*. The *krb.conf* file is used to specify the Kerberos realm and locations of the Kerberos servers. It also specifies the time to live of the ticket. The file is as follows

```
ticket_time 6000  
realm_name UMT.EDU  
UMT.EDU={  
admin_server darkstar.umd.edu  
kdc darkstar.umd.edu  
realm UMT.EDU }
```

The build is performed using standard make tools. After configuring all the files, the *Makefile* is executed which should produce all the executables required for the Kerberos system.

The *kdb5\_edit* program is used to add new users to the database as follows.

```
kdb5_edit
```

which prompts for the user name and the password. This password is further used while obtaining tickets. This password is encrypted using some crypt programs. Also the data is encrypted while transferring from client to servers.

The *kinit* is used to obtain credentials in order to communicate with the key distribution server as follows.

```
kinit username
```

Then it prompts for the password and the ticket is issued only when the password is validated.

The *klist* program is used to list the contents of the ticket file as follows.

```
klist
```