Firewall Monitoring

Jianyu You
NEC Systems Laboratory, Irving, Texas, USA
Ernst L. Leiss
Department of Computer Science, University of Houston, Houston, Texas, USA

Abstract

Securing resources against unauthorized access and/or use is a major concern of every organization that uses computer networks. To protect internal networks from external attacks, firewalls are utilized since they restrict network access while letting legitimate users have unencumbered access. Firewalls are also used to log security auditing information about connections and operations. We describe a monitor database gateway (MDBG) designed to replace older forms of firewall logging by a database system. SQL commands can be used to retrieve logged information instead of ad hoc scripts. The db application and allows secure access from other components of a firewall through the Kerberos authentication as well as other authentication methods. If the underlying database changes, only a small portion of the MDBG must be modified; the code for the other components of the firewall remains unaffected.

1. Introduction

The recent explosive expansion of computer networks has increased the dependence of organizations and individuals on the information stored and communicated using network systems, leading to a heightened awareness of the need to protect data and resources from unauthorized access and use, to guarantee the authenticity of data and messages, and to protect systems from network-based attacks. The types of attacks on the security of a computer system or network can be characterized by viewing the function of the computer system as providing information. Information flows from a source (a file, a region of main memory) to a destination (another file or user). We distinguish the following four attacks [15].

- Interruption: An asset of the system is destroyed or becomes unavailable or unusable.
- Interception: An unauthorized party gains access to an asset.
- Modification: An unauthorized party tampers with, an asset.
- Fabrication: An authorized party inserts counterfeit objects into the system.

A useful categorization of attacks is in terms of passive and active attacks:

- Passive Attacks: Eavesdropping on, or monitoring of, transmissions, to obtain information being transmitted.
- Active attacks: Modification of the data stream or the creation of a false stream.

Other attack taxonomies are discussed in [1].

To protect its systems from attacks, an organization must have security mechanisms and security services. Underlying most of the existing security mechanisms is cryptography. Very important in achieving these goals are firewalls, software/hardware systems that protect a computer network by enforcing an access control policy between two networks (see [2,13,14,17,20,21,22]). Usually, the network is protected against threats coming from an external network that cannot be trusted and from which security intrusions can originate. Protecting the network involves preventing unauthorized users from having access to sensitive data, while allowing legitimate users to have unencumbered access to the network resources.

A necessary part of a firewall is a bastion host. This is any firewall host exposed to an external network that forwards data between the internal and external networks. It is the central host in an organization's network security and must be closely monitored by the network administrators. Also important in TCP/IP (Transmission Control Protocol / Internet Protocol) networks are multi-homed hosts, hosts that have multiple network interface boards. Such hosts can route traffic between the connected network segments. A dual-homed host is a special case of a multi-homed host that has two network interfaces and has the routing functions disabled (Figure 1). A dual-homed host can be used to isolate an internal network from an external network. It is an example of a bastion host (Figure 2).

Firewalls are classified into three main categories ([2]): packet filtering, circuit gateways, and application gateways. A packet filter usually comes with screening router software, to connect to the Internet. Packet filters work by dropping packets based on

their source or destination addresses (ports). The administrator can define a list of the acceptable machines and services (based on port numbers) and a stoplist of unacceptable machines or services. Because a packet filter is able to analyze headers of the Internet's protocols (TCP/IP and UDP), TCP state flags can also be used to filter packets. UDP is connectionless and so does not retain state information. One must filter UDP based on port numbers.

Packet filtering technologies used in screening routers provide an efficient and general way to control network traffic. No changes are required to client and host applications (they operate on the IP and TCP layers), and these layers are independent of application-level issues. This simplicity has disadvantages:

(1) Because of the lack of context information, certain protocols such as UDP and RPC (Remote Procedure Call) cannot be filtered effectively. Also, auditing and alarm mechanisms are often missing.

(2) There are no good administration and user interfaces.

(3) The number of rules may be limited. Also, as the number of rules increases, the extra processing needed for every packet implies a high performance penalty.

(4) There is no way for the filter to distinguish securely one user from another.

Thus packet-filtering devices such as screening routers are often augmented by other devices called application-level gateways and circuit-level gateways (Figure 3).

Rather than using a general-purpose mechanism to allow many different kinds of traffic to flow, special purpose code is used for each desired application in **application-level gateways**. With them, one need not worry about interactions among different sets of filter rules, nor about holes in thousands of hosts offering nominally secure services to the outside. Only a chosen few programs need be scrutinized. Instead of a list of rules that control which packets or sessions should be allowed through, a program accepts the connection, typically performs strong authentication on the user which often requires one-time passwords, and then often prompts the user for information on what host to connect to. This is more limiting than packet filters and circuit-level gateways since there must be a gateway program for each application (e.g. telnet, ftp, X11, etc.). However, it provides much higher security because it can perform strong user authentication to ensure that the person on the other end of the connection is really who they say they are. Additionally, it can perform other access checks on a per-user basis (what times they can connect, what hosts they can connect to, what services they can use). Moreover, application-level gateways can be used to maintain an intelligent log of all usage of the applications. A major disadvantage of application-level gateways is that a custom program must be written for each application.

Circuit gateways relay TCP/UDP connections. The caller connects to a TCP/UDP port on the gateway which connects to some destination on the other side of the gateway. During the call, the gateway's relay program copies the bytes back and forth. In most cases, the connection service needs to be told the desired destination. A protocol describes the desired destination and service, and the gateway returns error information if appropriate. If the connection is successful, the protocol ends and the real bytes start flowing. These services require modifications to the calling program or its library. These gateways are also called "proxies". A very popular example is the SOCKS package, originally written by David Koblas [7], now maintained by NEC [23]. The SOCKS daemon runs on the firewall. Circuit-level gateways are more flexible and provide a general approach for building application gateways.

What Can a Firewall Not Do?

A firewall cannot control whatever happens after a user has passed the authentication and access checks. If a disgruntled employee properly identifies himself, properly logs into a secured machine, and then deletes files on that machine, the firewall did its job in ensuring that it was a properly authorized user.

A firewall cannot control people who go around it. That same disgruntled employee could plug his phone line into a modem attached to the network, dial up and bypass all firewall checks. For effective firewall use, network perimeter security integrity must be maintained. For more about host security, see [4].

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. Theoretically, virus detection is undecidable [9]: no general algorithm can exist to detect general viruses. In [9], virus examples, anatomy, diagnosis, prevention and cures are extensively discussed.

Reasons for Using a Monitor Database

An activity log of the connections through a firewall provides means for analyzing network security. By maintaining a log and reviewing it periodically, statistics about services under attack and other security violations can be obtained. One can observe trends and adjust security policies to tdeal with new threats. If one knows an external site had been penetrated and the hackers had collected passwords, one can warn the users of the system. If a userid is "root" or "sys" and failed to authenticate, it is probably an intent of break-in to gain control of the computer. Repeated failures during a short time interval are highly related with password-guessing attempts. Even for connection without errors, statistical anomalies in the user profile may signal possible attacks [9]. By keeping the database for a long time, it is possible to write scripts to detect carefully paced attacks over weeks or months.

Current practice logs all activities in a flat log file. This is inconvenient because the system administrators must write their own parsing code to analyze the log. A Monitor Database Gateway (MDBG) solves this problem by storing the log data into a database supporting SQL (Structured Query Language) queries. Retrieving connection data is much easier than from an unstructured log. Additionally, a monitor database can provide information related to billing, traffic analysis, and network usage.

2. The Monitor Database

The main purpose of the monitor database is to store data of network connections through a firewall. Based on client/server software, the programs use simple function calls like TCP networking, system time and DES encryption, so it can be ported easily to other Unix machines. Specifically, our approach offers the following advantages:

1. It lets other modules of the system work without changing their codes.
2. It provides secure data storage and retrieval.
3. It implements other features not provided by the underlying database like scheduling.
4. It supports more platforms in addition to that of the DBMS in use.

Consider the modules serviced by the Monitor Database Gateway (MDBG; Figure 4). All require access to the database. Without the MDBG, when the firewall is installed for a particular database, all these modules need to be rewritten in order to communicate with the underlying DBMS. The MDBG eliminates this necessity: when the firewall has to use a new DBMS, only a small fraction of code in the MDBG needs to be modified while authentication and networking code remain same.

A firewall system's own security is crucial, because the monitoring process can itself be subverted. Some intruders may be aware of the logging mechanisms. If the DBMS used for monitoring purpose has limited network security, the database gateway can add this capability.

2.1 Authentication

Authentication is the process of a computer reliably verifying the identity of someone, some process, or some computer.

Authentication functions

Message authentication can be viewed as consisting of two levels. At the lower level, there must be some function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message. The lower-level functions may be grouped into three classes ([15]):

(i) Message encryption: the encrypted text of the message serves as its authenticator.

(ii) Cryptographic checksum: A public function of the message and a secret key that produces a fixed-length values that serves as the authenticator.

(iii) Hash function: A public function that maps a message of any length into a fixed-length hash value, which serves as the authenticator.

Message encryption can be done using conventional or public-key encryption schemes. A widely used conventional or symmetric cryptosystems is the Data Encryption Standard (DES). Complementing one bit of the key or of the plaintext will flip approximately 50% of the output bits. Thus, almost no information about the input is leaked; small perturbations in the plaintext will produce massive variations in the ciphertext. A widely used cryptographic checksum, referred to as the Data Authentication Algorithm, is based on the DES [15].

Public key cryptography is also referred to as asymmetric cryptography. It was first proposed in 1975 by Diffie and Hellman ([15]). In conventional cryptography, it is impossible to communicate with two parties without prior contact. Additionally, the number of keys needed for a complete communication mesh is very large, $O(n^2)$ keys for an n-party network. Unlike in conventional cryptography, keys are not shared. Instead, each individual generates two keys: a private key that need not be revealed to anyone, and a public key that is preferably known to the entire world.

The best known public key system is RS an ([8,15]. If A wishes to send a message to B, it encrypts the message using B's public key. When B receives the message, it decrypts it using B's private key. No other recipient can decrypt the message because only B knows B's private key. If A wants to authenticate B, A can send to B in cleartext a random number, e.g., the current time of day as known to A. B encrypts the received time of day using its own private key and sends this to A. Now A decrypts B's message using B's public key and compares it with the time when A received the message. If the delay is short, A authenticates B's identity because no one else can encrypt the time of day such that when the ciphertext is decrypted with B's public key, it yields to the cleartext time of day. If A wants to sign a message and send it to B, A applies the decryption function to the cleartext message with its private key and encrypts the result with B's public key to produce a ciphertext. B can decrypt the ciphertext using its private key, then encrypt the result using A's public key to reproduce the original message. By saving the ciphertext, B can actually supply a proof that A signed the message.

Another way to authenticate a message is using a secret key to generate a small fixed-size block of data, a cryptographic checksu, or message authentication code (MAC), that is appended to the message. This technique assumes the two parties share common secret keys. In the basic scheme, the sender computes the cryptographic checksum and the message plus checksum are transmitted to the receiver. The recipient performs the same calculation on the received message, using the same secret key, to generate a new cryptographic checksum which he compares to the received checksum. Assume that only the sender and receiver know the secret key; if the received checksum matches the computed checksum, then:

1. The receiver is assured that the message has not been altered. Because the attacker is assumed not to know the secret key, he cannot alter the checksum to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender.
3. If the message includes a sequence number (such as implemented in the MDBG), the receiver can be assured of the proper sequence, because an attacker cannot successfully alter the sequence number.

Notice that a cryptographic checksum provides authentication but not confidentiality. In many contexts, messages are not secret but their integrity must be assured. In such contexts it is perfectly appropriate to transmit unencrypted data, plus an MAC.

It is often impractical to apply encryption to an entire message. Instead, we may use a secure hash function. A secure hash function must have two properties. First, its output is short, on the order of 128 bits. Second, and more important, it must be infeasible to create an input that will produce an arbitrary output value. Thus, an attacker cannot create a fraudulent message that is authenticated by means of an intercepted genuine hash value. Well-known hash functions are MD2, MD4, MD5 and SHA ([6,15]). A common way to use the hash function for authentication is to let the sender append to the message a secret key known

to the receiver and to compute the hash value of the whole string. The receiver does the same with the message and the secret key. If the computed hash value matches the received one, the receiver authenticates the sender because nobody else can produce the hash value without knowing the password.

Authentication protocols

Password-based authentication refers to a secret quantity (the password) that a user states to prove he knows it. So, the user (or a host) sends his name and password in cleartext through the communication channel. The problem with password-based authentication is eavesdropping. Many protocols were designed for an environment where eavesdropping was not a concern (rightly or wrongly), and attackers were (again rightly or wrongly) not expected to be very sophisticated. It is also used when a cryptographic algorithm is not available or expensive to perform.

Address-based authentication does not rely on sending passwords over the network, but rather assumes that the identity of the source can be inferred based on the network address from which packets arrive. This technique is safe from eavesdropping, but is subject to two other threats: a) An attacker can gain access to a node, which is authorized to use other resources. b) An attacker can impersonate the network address of trusted nodes. Depending on the environment, therefore, address-based authentication may be more or less secure than sending passwords in the clear.

Cryptographic authentication protocols can be much more secure than either password or address-based authentication. The basic idea is that a user proves his identify by performing a cryptographic operation on a quantity known by the server. The cryptographic operation performed by the user is based on his secret password. The password can be used to acquired a cryptographic key either directly, as in doing a hash of the password, or using the password to decrypt a higher-quality key, such as an RSA private key, that is stored in some place like a directory service.

Trusted Intermediaries is another efficient protocol when many entities are involved in the authentication. Assume that network security is based on secret key schemes. If the network has n nodes, each computer must know n-1 keys, one for each other system on the network. Adding a new node requires n keys to be generated and securely distributed. This is clearly unworkable in large networks and is avoided by using a trusted node known as a Key Distribution Center (KDC). If a new node is installed in the network, only that new node and the KDC need to be configured with a key for that node. If a computer wants to talk to another computer, KDC can help them authenticate each other. Kerberos is a well-known trusted intermediary which will be discussed in more detail below.

Normal authentication in the MDBG

In our MDBG, users are divided into two groups: monitor users and GUI users. Monitor users have the privileges of inserting and updating the connection data, so a strict authentication is enforced. GUI users have fewer privileges than monitor users; e. g., they cannot insert or update the connection data in the files. So their authentication can be weaker than that of the monitor users.

Password-based authentication is used when a GUI user identifies itself to the MDBG. This is adequate, because GUI users do not alter the database, and the database is accessed from the internal network (internal users are trusted). For monitor users, encryption based authentication is utilized. Each monitor user sets up a password on both server and client machines. The monitor program encrypts the timestamp and its IP address and sends them to the server which decrypts the information and checks whether the timestamp has expired and whether the IP address coincides with the source of the message.

Address-based authentication is performed for both monitor users and GUI users, to ensure that accesses to the MDBG come from the internal network. After the encrypted data are analyzed, the server checks if the IP address is in the database host file. If this succeeds, the server sends a packet for recognition. Hereafter, the server receives SQL commands from the client and returns answers.

The technique used in the MDBG to authenticate individual messages is based on a cryptographic checksum. After the output of the encrypted checksum is computed, 32 bits are taken along with 4 bytes of sequence numbers to form a block. Then these 64 bits are encoded as the MAC (Message Authentication Code) and sent to the server. On the sender end, the checksum is compared with that computed by the server and the sequence number is verified. By using this method, there is no realistic way that an attacker can replay the message since it is very difficult to alter the message. The chance that an attacker uses another message with identical checksum is only 2^{-32}.

2.2 Kerberos Authentication

Kerberos [16] has become the de facto standard for authentication in networking. It serves two purposes: authentication and key distribution. Each user and each service share a secret key with the Kerberos key distribution center. Kerberos is a trusted third-party authentication service using a protocol based on that proposed by Needham and Schroeder. Like traditional authentication methods, Kerberos relies on secret passwords to ensure the identity of a person or a machine on the network. Unlike traditional methods, Kerberos has several features for safeguarding the secrecy of the passwords. It provides a protocol which facilitates mutual authentication and which never sends passwords over the network in clear text. Also, it stores passwords only in secure Key Distribution Centers (KDC). Kerberos allows the use of encryption to secure the integrity of data sent over the network and to prevent eavesdropping.

Any user, application, or host system known to Kerberos is a principal. To use Kerberos, a user must first run the program kinit, which identifies the user to the Authentication Server (AS) that runs on the KDC. The AS returns a ticket granting ticket (TGT); it allows the user to obtain tickets to communicate with other kerberized services on the network. The TGT is encrypted using a secret key that is derived from the user's password. kinit prompts the user for the password which it transforms into the user's secret key and uses it to decrypt the contents of the TGT, including a session key which must be used for subsequent communication with the AS. The TGT and the session key are saved in the user's credential cache. Note that the user's password is never sent over the network, neither encrypted nor in cleartext.

To request a service under Kerberos, a ticket for that specific service must be obtained (Figure 5). This is done automatically by a kerberized client program for that service. The client program retrieves the TGT from the user's credential cache and sends it along with the specific service request to the Kerberos Ticket Granting Service (TGS) running on the KDC. The TGS sends the ticket for the requested service to the client program, which stores the service ticket in the user's credential cache and also presents it to the application server for authentication. Part of the ticket is encrypted using a secret key shared only by that application server and the KDC. If the server successfully decrypts that portion of the ticket, it trusts that the ticket was issued by Kerberos; the user is authenticated and the requested service will be performed.

Authentication across a realm (a collection of principals under a common Kerberos administration) boundary is also possible. An ordinary ticket will not suffice, since the local KDC will not have a secret key for each and every remote server. Instead, an inter-realm authentication mechanism is used. The local KDC must share a secret key with the remote server's KDC; this key is used to sign the local request, thus attesting to the remote KDC that the local one believes the authentication information. The remote KDC uses this information to construct a ticket for use on one of its servers.

Advantages of Kerberos

Hard to break: Since both the ticket and authenticator contain the network address of the client, another workstation can't use stolen copies of these without changing their system to impersonate the owner's network address. Also, an authenticator has a short lifetime and is only valid once.

Password protection: Passwords are never sent across the network in cleartext. They are always encrypted. Additionally, passwords are never stored on a workstation or server in cleartext.

Transparent: Ideally, the user is not aware that authentication is taking place, beyond the requirement to enter a password. The user only has to identify himself once, at the beginning of a workstation session.

Low cost: It available from MIT effectively for free and includes publicly usable encryption algorithms.
Reliable: The code is fairly mature. It is widely used and tested.

Drawbacks of Kerberos

Although Kerberos is better than the address-based authentication methods used in earlier protocols, it does have some weakness and limitations [2,17].

Most computer systems do not have a secure area where to save keys. Because a plaintext key must be stored in the initial dialog to obtain a ticket-granting ticket, a secure place is needed to store it. If this plaintext key is compromised, the entire realm can be easily compromised.

The Kerberos system was developed at MIT as part of Project Athena. The disadvantage is that the environment at MIT is unlike any other organization. Kerberos was designed to authenticate the end user - the human being sitting at the keyboard - to some number of servers. It is not a peer-to-peer system, nor was it meant for one computer system dàemon to contact another computer.

Another problem is how Kerberos handles keys on multiuser computers. Cached keys can be obtained by other users logged in to the system. In a single-user work-station environment, only the current user has access to system resources, so there is little or no need to enable remote access to the workstation. However, if the workstation supports multiple users, it is possible for others to obtained the keys.

The KDC is a single point of failure. If it fails, nobody can use anything on the network. It is possible to have multiple KDCs which share the same database of keys, but this adds complexity and cost (extra machines and replication protocols) and vulnerability (more targets to be protected).

2.3 Use of Kerberos in the MDBG

The MDBG can authenticate a user or the monitor program running on other machines with Kerberos. During the configuration of the MDBG, if Kerberos is detected on the same machine, the MDBG will be compiled and linked with the Kerberos library [11]. A database API is implemented for the monitor program. It consists of three functions. When the monitor program runs and uses the monitor database API, db_open() is called before any other access. The encoded password saved in a password file is retrieved and several Kerberos functions are called to initialize the credential cache and build the principal data structure. Immediately before Kerberos obtains TGT, the password is decoded. This ensures that if some function call fails and the program exits, no cleartext password remains in the memory. Immediately after authentication call, the password is removed from the memory, no matter whether the TGT is successfully obtained. The usual lifetime of a valid TGT of 8 hours is reduced to 5 minutes.

When an MDBG user runs the interactive SQL client (isql) and uses Kerberos to identify himself to the DBMS server, the credential cache should have a valid ticket granting ticket. Otherwise kinit should be run in order to get a TGT. After obtaining the TGT, a ticket for the MDBG service must be obtained. This is done automatically by client programs that construct communicating principals, retrieve the TGT from the default user's credential cache, and send it along with the service request to the Kerberos Ticket Granting Service (TGS). The TGS sends back the ticket for the requested service to the client programs. The client programs store the service ticket in the user's credential cache and also present it to the application server for authentication. Part of the ticket is encrypted using a secret key shared only by that application server and the KDC. If the MDBG server can successfully decrypt that portion of the ticket, it can trust that the ticket was issued by the Kerberos. The server must always authenticate itself so that the clients can be sure that it is talking to the intended server and not an impostor.

Access Control

Authentication establishes who the user is. Authorization establishes what the user is allowed to do. Sometimes the user can prove authorization without divulging his identity. In most situations, especially those involving resource access across a network, security does involve the two steps of authentication (establishing who is) and determining his authorization.

There are two access control mechanisms: discretionary access control and mandatory access control [1,3]. A discretionary access control mechanism comprises those procedures that enforce the specified mediation at the discretion of individual users, a mandatory one comprises those procedures that enforce the specified mediation, not at the discretion of individual users, but rather at the discretion of a centralized administration facility. MDBG is based on Unix, so only discretionary access control is used.

An ACL (Access Control List) for each resource could be used for authorization; however, maintaining an ACL on every resource can quickly become prohibitively expensive. A common solution is to introduce the concept of groups. However, there is still some inefficiency as each system manager is maintaining his own ACL, duplicating effort. In our MDBG, access control is performed in two steps. The first step is through the list of authorized users maintained by the MDBG and the second step is through the database management system itself by maintaining database access privileges with GRANT/REVOKE commands in SQL. There are two possibilities:

1. If Kerberos authentication is used: After the MDBG authenticates a user, it goes to the password file to retrieve the encrypted password associated with the user. This password search serves a twofold purpose: (i) if the user is not in the password file, he is not an authorized user; the authorization fails; (ii) the encrypted password is decrypted and used to log into the DBMS.
2. If normal authentication is used: Authorized users are stored in the password file; to authenticate a user, the MDBG retrieves his password in the password file. If it is not found, the authorization fails. For an authorized user, the MDBG uses his name and password to log into the DBMS.

The second step of authorization is performed by the DBMS, with finer granularity. The DBMS has a rich set of grant/revoke commands to setup a particular privilege (connect, select, create, drop, etc.) on a particular object (table, view, column, etc.). When a monitor program creates the tables by API to store monthly connections, it grants only the "select" privilege to Public while retaining all privileges for itself.

The concept of groups is not used in the MDBG, since many DBMSs do not support this concept (Sybase does, Oracle and JLT do not). Instead, users are divided into two categories: monitor users and GUI users. Monitor users are those who own the databases and can grant or revoke other user's privileges. They can also set their process's priority. GUI users can only read the database and access other resources if a monitor user grants them some additional privileges.

Priority setting

The database gateway allows user to define a priority scheduling. In fact, the database can provide real-time capabilities. Our priority setting is a rather simple illustration of this capability. Consider the following situation: The system administrator receives an alarm from the firewall and suspects an intrusion is underway. He wants to know past history of the user or host. At this moment, he issues a query to the database. Evidently, it is desirable that the database answers as quickly as possible in order to let the administrator make a correct decision. However, in this moment, some GUI user may issue a long query; thus, GUI user and system administrator will share the processor, and the answer to the administrator's query is slowed down. The system administrator should be able to use a higher priority to speed up his queries. A real-time intrusion detection system using a database is described in [10].

CPU scheduling in Unix is designed for interactive use. Processes are given small CPU time slices by a priority algorithm that reduces to round-robin scheduling for CPU-bound jobs. Every process has a scheduling priority associated with it; large numbers indicate lower priority. Processes doing disk I/O or other important tasks have negative priorities and cannot be killed by signals. Ordinary user processes have positive priorities and thus are less likely to be run than are any system process, although user processes may set precedence over one another through the "nice" command. Process aging is employed to prevent starvation, even if there is no preemption of one process by another. Since the operating system has a reliable scheduling algorithm, the MDBG uses the second approach, that is, it assigns a process priority and lets the operating system do the scheduling.

When GUI users request DBMS service, they cannot assign their own priorities; the MDBG always assigns them a priority of 0. When a monitor user requests the service, he may specify a desired priority since monitor users are trusted and the MDBG is designed to let them assign their own priorities.

Password setting

The monitor database gateway allows users to connect to the database system without going through the normal system authentication. It is important to authenticate all users. One may want to let the MDBG server have root privilege to have access to the system's password file. But if it is running on a big machine with some DBMS like Oracle or Sybase, it is unlikely the system administrator will let untested software run with the root privilege. Instead the MDBG server can login on the DBMS machine requesting user's username and password. If it does not have access to the system's password file, it may have its own password file to store (hashed) passwords. For more details and discussion, see [19].

Monitor database server

Figure 6 outlines the functional description of the MDBG server. The server allows users to specify some settings, different from default values like port number. It starts up as a daemon process and sits on a well known port (currently 6548). Upon a service request, it forks a child process to handle the SQL command, then the parent process waits for another request to arrive. When a query result is obtained, the server formats it in packets and sends them to the client. On the client side, minimal processing is required to display the data.

2.4 SQL Processing

The essential part of the MDBG server is the SQL processing. Separate modules for supported DBMSs (Oracle, Sybase, Just Logic Technology) are implemented and are described below.

Oracle: Oracle is the world's largest vendor of software for managing information with about 50% of the market share for mainframe databases. The normal programming tool to write a user application (not data entry interface, in the latter case, SQLFORMS is used) is embedded SQL for different languages; PRO*C is chosen for our implementation. In embedded SQL ([12]), before executing any SQL command, we distinguish two kinds of commands, without (non-select statement) or with output (select statement). Examples are: insert a record into the database, commit a transaction, etc.

Sybase: Sybase is the second largest database vendor. It leads the industry in client/server computing. The programming tools for user applications are Open Client Library function calls or embedded SQL. Programming with embedded SQL is similar to that of Oracle and JLT (described later). The Open Client Library is more widely available for users who use Sybase because Sybase's embedded SQL translates SQL to Open Client Library calls. Here, the MDBG is implemented with the Open Client Library ([18]). Open Client makes little distinction between select statements and nonselect statements.

Just Logic Technology: Just Logic Technology DBMS [5] is a small relational database management system. The system is cheap (about $300 for stand-alone BSDI, about $400 for client/server BSDI) and can be installed on a BSDI, SCO or Linux based PC. JLT gives users the choice of three programming interfaces to suit their own needs [5]. Because of portability, embedded SQL is chosen for the MDBG.

2.5 Monitor Database Clients and APIs

An SQL client allows a user to type in SQL commands on a remote machine and get result displayed on his screen. It allows users to specify some settings, different from the default values like hostname, port number, priority and program trace, to identify themselves to the server, to perform simple command parse and to send commands to the server, to receive query results from the server and display them neatly, and to provide some help to the user.

3. Applications of the Monitor Database

There are two ways for a firewall to store data in a monitor database. The first is to let each proxy of the firewall send data to the MDBG. An application of this approach is illustrated in [19] with the popular proxy SOCKS. The second is to implement a program (e.g., a monitor) to collect data from the proxies and to send data to the MDBG. This application is illustrated using an

NEC firewall in [19]. One of the benefits of logging network operations is statistical analysis. A graphic user interface (GUI) for network use statistics is described in [19]. This GUI retrieves data from the MDBG and displays them graphically to the user in a variety of ways.

4. A Firewall Access Definition Language

Currently, many firewalls use a list format to define the access control list. For example, SOCKS v4 [23] defines the following format for access control:

action [*=userlist] src_addr src_mask [dst_addr dst_mask] [op dst_port] [: shell_cmd]

Thus

permit *=boss,root 1.2.3.4 255.255.255.255 11.12.13.14 255.255.255.255 eq telnet

means that the SOCKS server allows requests from a local user whose effective id is either boss or root, the source IP address must be 1.2.3.4 exactly, the destination IP address must be 11.12.13.14 exactly, and the service requested must be telnet. When fields are numerous (including source and destination's IP addresses, masks, protocols, port numbers, users, access time, authentication methods, etc.), defining the access control lists is error prone and inconvenient.

Our Firewall Access Definition Language (FADL) aims at providing sequential statements familiar to system administrators to generate the access control list. Though the grammar is designed to be simple at the beginning, the language construct allows expansion to meet future requirements. Using a language for defining a firewall's access control list permits a clear description of the access through natural IF..THEN..ELSE statements. It is easier to use; e. g., application names can be used instead of specifying protocol and port numbers. It is flexible: if an additional variable has to be taken into consideration, we can define a global variable instead of adding a field to every line in the normal ACL.

In our FADL, the configuration file of a firewall under consideration has two sections, (1) Connection permission, and (2) Alarm specification. The FADL consists of only three language constructs to define these sections: assignment statement, conditional statement and alarm specification. In [19], a description of the FADL is presented, then the grammar of the language is given. A method for checking consistency among the statements is developed and the combination with alarm specifications using the FADL language construct is discussed.

5. Conclusions and Extensions

A system to provide secure data storage for a firewall has been implemented. A difficult part has been finding a way to accommodate the system for different behaviors of the DBMSs. Thus, the semantics of SQL commands should be unified. The authentication scheme, the server program, and the network functions implemented in this thesis can be reused. Only the creation of new tables and the use of appropriate SQL commands to store data needs to be changed. Another extension would be to add more capabilities to the monitor database server, for instance, a better real-time capability or the processing of application specific commands, because if the client program needs to obtain the statistics for a time interval crossing different months, several SQL command must be issued to retrieve data corresponding to each month and to consolidate the total result.

6. References

[1] E. Amoroso, *Fundamentals of Computer Security*, Prentice Hall, 1994.
[2] W. R. Cheswick and S. Bellovin, *Firewalls and Internet Security*, Addison Wesley, 1994.
[3] K. Bhaskar, *Computer Security: Threats and Countermeasures*, NCC Blackwell, 1993
[4] S. Garfinkel and G. Spafford, *Practical Unix Security*, O'Reilly and Associates, 1991
[5] Just Logic Technology, Just Logic/SQL Database Manager, (user manual), Just Logic Technology, 1995.
[6] C. Kaufman, R. Perlman, and M. Speciner, *Network Security: PRIVATE Communication in a PUBLIC World*, Prentice Hall, 1995.
[7] D. Koblas and M. Koblas, "SOCKS", Proc. 1992 Usenix Security Symp., 77-83, 1992.
[8] E. L. Leiss, *Principles of Data Security*, Plenum, 1982.
[9] E. L. Leiss, *Software Under Siege: Viruses and Worms*, Elsevier, 1990.

[10] T. F. Lunt and R. Jagannathan, "A Prototype Real-Time Intrusion-Detection Expert System", Proceedings of IEEE Symp. on Security and Privacy, 59-66, 1988.
[11] MIT Information Systems, Kerberos V5 Application Programming Library, MIT Information Systems, 1994.
[12] Oracle Corporation, Pro*C Supplement to the ORACLE Precompilers Guide, Oracle Corporation, 1990.
[13] M. J. Ranum, "An Internet Firewall", Proc. World Conf. Systems Management and Security, 1992. Available by ftp from decuac.dec.com:/pub/docs/firewall/firewall.ps.
[14] M. J. Ranum, "Thinking About Firewalls", Proc. Second World Conf. Systems Management and Security, 1993. (http: www.tis.com/Home/NetworkSecurity/Firewalls/ThinkingFirewalls.html.
[15] W. Stallings, *Network and Internetwork Security*, Prentice Hall, 1995.
[16] J. Steiner, C. Neuman, and J. Schiller, "Kerberos: An Authentication Service for Open Networked Systems", Proc. Winter 1988 USENIX Conference, 191-202, February 1988.
[17] K. Siyan and C. Hare, *Internet Firewalls and Network Security*, New Riders Publishing, 1995.
[18] Sybase Inc., Open Client DB-Library, Sybase Inc., 1991
[19] J. You, " Firewall Monitoring Using Databases", M. S. thesis, Dep. Computer Science, Univ.of Houston, Houston, Texas, December 1995.
[20] ftp://ftp.greatcircle.com - Firewalls mailing list archives, directory: /pub/firewalls
[21] ftp://ftp.tis.com - Internet firewall toolkit and papers, directory: /pub/firewalls
[22] ftp://research.att.com - Papers on firewalls and breakins, directory: /dist/internet_security
[23] http://www.socks.nec.com - Document for SOCKS and free software, directory: /

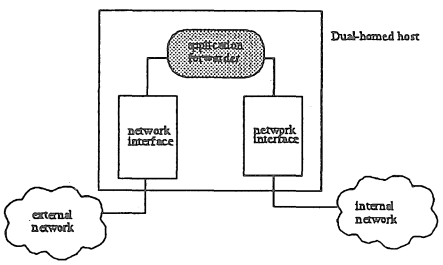

Figure 1 Dual-homed host

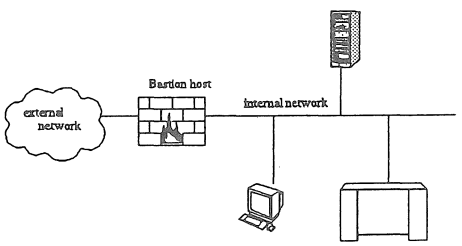

Figure 2 A Bastion host (dual-homed)

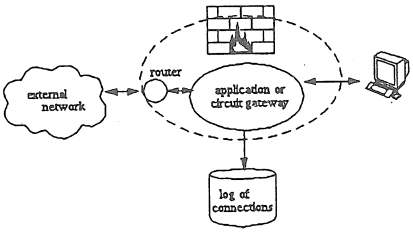

Figure 3 Application and circuit gateway

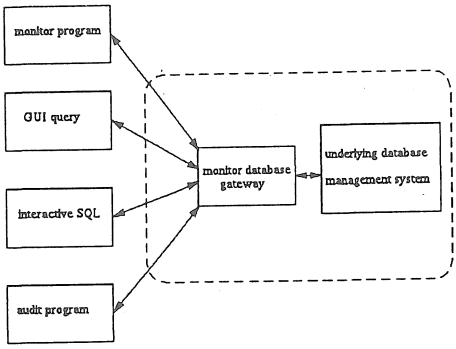

Figure 4 . Modules serviced by the MDBG

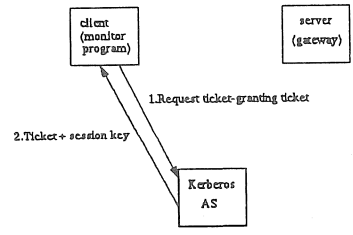

Figure 2.4 Process of getting TGT during db_open (equivalent to typing kinit)

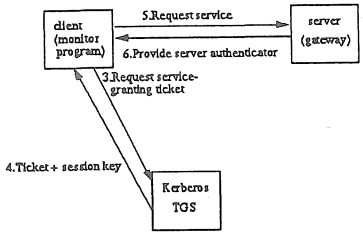

Figure 5. Obtaining a service

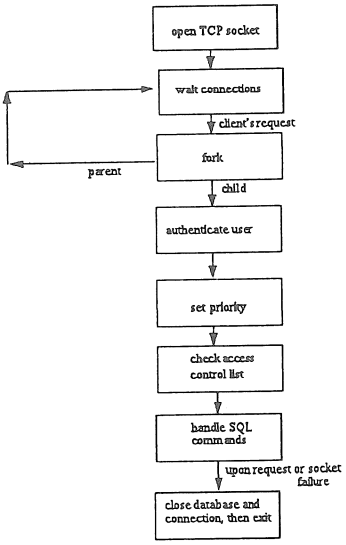

Figure 6 Steps performed by the MDBG server