SANDIA REPORT

SAND96-2344 • UC-705
Unlimited Release
Printed September 1996

Information Integrity and Privacy for Computerized Medical Patient Records

Joselyne Gallegos, Victoria Hamilton, Timothy Gaylor, Kevin McCurley, Timothy Meeks

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

SF2900Q(8-81)

SAND96-2344
Unlimited Release
Printed September 1996

Distribution
Category UC-705

Information Integrity and Privacy for Computerized Medical Patient Records

Joselyne Gallegos and Victoria Hamilton
Data Systems Security Department

Timothy Gaylor
Data Transport and Network Design Department

Kevin McCurley
Parallel Computing Sciences Department

Timothy Meeks
Software Integration, Technology, and Standards Department

Sandia National Laboratories
Albuquerque, NM 87185

Abstract

Sandia National Laboratories and Oceania, Inc. entered into a Cooperative Research and Development Agreement (CRADA) in November 1993 to provide "Information Integrity and Privacy for Computerized Medical Patient Records" (CRADA No. SC93/01183). The main objective of the project was to develop information protection methods that are appropriate for databases of patient records in health information systems. This document describes the findings and alternative solutions that resulted from this CRADA.

C/SNL--SC9301183

Contents

Figures

Tables

Chapter 1 - Introduction

Sandia National Laboratories and Oceania, Inc. entered into a Cooperative Research and Development Agreement (CRADA) in November 1993 to provide "Information Integrity and Privacy for Computerized Medical Patient Records" (CRADA No. SC93/01183). The main objective of the project was to develop information protection methods that are appropriate for databases of patient records in health information systems.

1. Purpose

The purpose of this document is to describe the findings and alternative solutions that resulted from this CRADA. This paper documents products and technologies which may be useful to Oceania's development effort. However, the authors should not be assumed to endorsing particular products. Any product information is purely informational. Other products may be equally satisfactory.

It should also be noted that the versions of Oceania software, including database definitions and application software, possessed by the authors were obtained approximately one year prior to the publishing of this report. The findings and alternative solutions presented are based on these versions. Oceania has since proceeded on an aggressive development schedule leading to similar findings and solutions.

2. Overview

Chapter 2 includes a description of the basic techniques and technologies used to provide information surety. Chapter 3 details mechanisms for securing a UNIX database server including specifying UNIX functionality which should, if possible, be avoided. Chapter 4 provides information useful in securing the database itself. It includes several detailed examples discussing database audit trail services. Chapter 5 describes the results of investigation into the current capabilities of smart cards. Chapter 6 summarizes the previous chapters by detailing system architecture options and their associated information surety features. The document includes several appendices. Appendix A is a tutorial on multi-tiered architectures. Appendix B and C describe the Distributed Computing Environment and the Open Horizons Connection products respectively. A complete list of references, including web pages follows. Finally, the definitions of technical terms and phrases used in this specification are given in the Glossary. Acronyms and abbreviations also appear in the Glossary.

Chapter 2 - Basics of Information Surety

Sandia National Laboratories uses the term ***information surety*** to refer to balancing confidentiality (or privacy), integrity, and availability of data. The term information surety refers to a balance between protection against unauthorized use of information and assurance of authorized use. By contrast, the term ***information security*** connotes an emphasis on protection against unauthorized use of information.

The information surety issues that were addressed in this project were to:

- determine whether the user of a system is who he claims to be;
- ensure that data maintains its privacy;
- ensure that data is not changed or accessed by an unauthorized action or user;
- ensure that data is not changed by accidental means;
- ensure that data cannot be refuted;
- determine when data was created or modified;
- determine who created or modified data;
- determine when data was accessed;
- determine who was granted what access by the system; and
- ensure that data will be available and useable when needed.

The following techniques and technologies were assessed and integrated into the design of several security infrastructures, to address the information surety issues listed above.

- Encryption
- Strong Authentication
- Access Control
- Digital Signatures
- Smart Cards
- Secure Timestamps
- Audit Trails
- Key Management

3. Threat Analysis

Before designing surety into any system, it is important to know about the potential threats so that the appropriate counter-measures can be taken. It is impossible to predict every threat to a system, and it is very expensive to protect against all threats. Threats must be prioritized and weighted according to their seriousness and consequences. A balance must be struck between the cost of surety and the value of the assets being protected. In essence, providing surety is a matter of risk management.

In addition to malicious threats against the surety of a system, the threats resulting from hardware, software, and network failures must also be considered. This makes the case for formal analysis, design, implementation, testing, and verification of systems even more compelling. Human errors often stemming from ignorance are also threats to the surety of a system. Users must be educated about information surety. Usage policies and security policies help prevent security incidents and provide guidance for when incidents do occur. The policies must include guidelines for violations and they must be enforced. Users must be made aware as to the appropriate uses of resources.

4. Encryption

Encryption is a method of making information indecipherable [Schneier, 1996] to authorized users. The purpose of encryption is to protect data from unauthorized viewing or use during transmission and storage. Encryption algorithms generally require at least one key and may require more than one key. A good encryption algorithm is one that has the property that even if the algorithm is known, data may not be deciphered without the key(s). The length of the key(s) also affects the strength of the algorithm. Symmetric key encryption requires that both the originator and intended recipient use a shared key. Asymmetric key or public-key encryption requires that the originator use the publicized key of the intended recipient to encrypt and the intended recipients use his/her private key to decrypt. Data Encryption Standard (DES) is an example of a widely available symmetric key algorithm [NISTa, 1993]. The Rivest, Shamir, Adleman (RSA) algorithm is an example of a widely available asymmetric key algorithm [Rivest, 1978].

There are several issues associated with the use of encryption technology. Since encryption generally requires the use of keys, key management must be addressed (see the Key Management section for further discussion of these issues). In addition, the export of encryption software and hardware is closely controlled by the United States government. Encryption methods that use keys greater than a given length (generally 40 bits for most symmetric algorithms) are subject to scrutiny and fairly rigorous export control [DOS, 1989, 1992].

5. User Authentication and Network Authentication

Identification of legitimate users is a prerequisite to any sort of security. In a networked system in which data is passed between machines, identification is also required between different computers on a network.

It is generally acknowledged that user authentication can be accomplished by using some combination of something known by the user (e.g., a password or PIN), something possessed by the user (e.g., a badge or token), and something unique to the user (e.g., a fingerprint). Two-factor authentication is built by a combination of at least two of these, such as a password and a token. In this case, an adversary must have both the password and the token to impersonate a user. Strong user authentication is the establishment of validity of a claimed identity using cryptographic techniques.

In today's client/server systems, communication will need to take place between a client workstation and the server system where the data actually resides. Users will need to authenticate themselves not only to the server system but also to the client system. The major problem in this scenario is how to transmit the authenticator (most commonly a password or passphrase) over open networks from the client to the server. One system to designed to address this problem is Kerberos, which was developed at M.I.T. [Neuman, 1993].

6. Access Control

Access Control refers to the ability of the system to grant, revoke, determine, and enforce different privileges granted to users within the system. Access control and user authorization mechanisms are required to permit or deny users the abilities to read from, write to, and execute particular system resources. For instance, access to computer systems in general is usually provided through the use of unique user identifiers, passwords, group memberships, etc. Access to computer files is provided through privileges granted to individual user identifiers and groups of users. Access to database objects, tables, etc., is also controlled through the use of privileges granted to similiar subjects (i.e., user identifiers and group of users).

7. Digital Signatures

A digital signature algorithm is a public-key cryptographic method used by the data's recipient and any third party to verify the identity of the originator of the data. It can also be used to verify the authenticity of the data. An originator of data creates a digital signature by uniquely transforming the data with his or her private key. A recipient, using the originator's public key, verifies the digital signature by applying a corresponding transformation to the data and the signature [Schneier, 1996]. Digital signatures may also be used to support non-repudiation. Non-repudiation means that the originator cannot at a later date deny data origination, modification, or deletion. The Digital Signature Algorithm (DSA) [NIST, 1993b, 1994] and RSA [Rivest, 1978] are two of the most common digital signature algorithms.

Since digital signature algorithms require the use of keys, key management must be addressed (see the Key Management section below). Unless a digital signature algorithm can be used for encryption as is the case with RSA, its export is not controlled by the United States government. However, developers should verify this before attempting export.

8. Smart Cards

ISO-standard smart cards are similar in form factor to standard credit cards, but they have an embedded microprocessor and I/O channel. There are different types of smart cards with different capabilities. See the chapter on Smart Cards for a more detailed discussion. For this project, the smart cards considered are those with cryptographic capabilities. Such smart cards can be used in key management, strong user authentication, encryption, and digital signature generation.

9. Audit Trails

Audit trails are one of the most important security mechanisms. When implemented properly and securely, these mechanisms allow the tracking of ***who*** accessed ***what*** information in the system and ***when***. They are often instrumental in implementing non-repudiation requirements. Note that access to information includes reading, appending, generating, modifying, or deleting information. This mechanism could be used in court, if reliable enough, as evidence of proper (or improper) handling of health information. In fact, the very existence of audit trails often serves as a deterrent for the mishandling of information. The audit trail is an appealing technology since it minimizes reliance on trusted users. Today's systems require that audit trail data be accessible to system administrators and system security administrators, who are generally considered trusted users. Mechanisms must still be developed to protect the audit trail from abuse by these users. Legal may determine how long audit data must be kept. Audit tools may be necessary for convenient viewing of audit data.

Access logs can be constructed from audit trails. Whenever a part of a system or resource is accessed, what information was accessed, who accessed it, and for what purpose can be logged. If this is the case, then protection from unauthorized access to these logs must be deemed important. Access patterns could themselves convey sensitive information.

10. Secure Timestamping

In addition to protecting the content of information in electronic records, it is also necessary to record the time at which an entry was made in the record so that records cannot be changed later. If the date of an action is simply entered into the record, then this date can later be changed like any other piece of information, unless proper cryptographic techniques are applied to timestamps to prevent such changes. This is particularly important for the use of electronic records in legal environments (for example, to satisfy non-repudiation requirements).

Other applications for cryptographically secure timestamps are audit trails and digital signatures. In audit trails, it is necessary to record the time at which an entry was made in the audit trail so that the audit trail itself cannot be changed later. For digital signatures to truly provide non-repudiation, timestamp information must be included. One of the reasons for this is that if a user chooses to repudiate his signature (e.g., his private key was compromised), then he should not have complete freedom to repudiate individual signatures. If his key was indeed compromised, then it may be necessary to issue a revocation of signatures generated after that date. Another reason to include secure timestamps is that they can be used to detect replay attacks. A replay attack occurs when an adversary intercepts a valid message and transmits it at some later date. In addition, digital signatures, or more accurately the public key certificates associated with digital signatures, also have expiration dates.

11. Key Management

Key management is the process of:

- generating keys, including the choice of appropriate key values;
- protecting keys within the system;
- verifying the authenticity of keys;
- using the keys;
- storing the keys, including escrow and archival;
- changing the keys;
- destroying the keys; and
- handling compromised keys.

Any system which makes use of cryptographic keys must address key management. Key management can often be the bulk of a secure system focus. In addition, any system which employs public-key or asymmetric algorithms for encryption or digital signatures must provide a mechanism for linking a user identity to his or her public key. One mechanism for accomplishing this connection is the implementation of certificates. Certification as defined in the X.509 standard [ITU-T, 1993] is accomplished by a certification authority or a chain of certification authorities that are trusted by users. Users obtain certified public keys from a certification authority or are notified that their request cannot be met if the public key has been revoked or has expired.

12. Policies

Usage policies and security policies help prevent security incidents and provide guidance for when incidents do occur. The policies must include guidelines for violations and they must be enforced. Users must be made aware as to the appropriate uses of resources. In the health care environment, guidelines for the handling of information must be designed with an eye towards government legislation and regulations.

Chapter 3 - Securing a UNIX Database Server

1. Overview and Goals

The purpose of this chapter is to provide an overview of how to secure a SQL server running on top of a generic UNIX platform, as is required for the Oceania CRADA. This chapter is aimed at a system administrator who is familiar with basics of UNIX, and does not address a particular vendor's operating system. A system that is configured according to the principles here should be quite resistant to all but the most dedicated hackers, but each variation of UNIX will have its own peculiarities. It is recommended that the vendor's instructions for patching security holes be used to complement the procedures discussed here. Some vendors are more responsive than others in patching such holes, but the increasing reliance on the Internet has made all of the vendors more conscious of security problems.

Providing security for a *complete* information system will require attention to other aspects not covered here, including:

- key management for encryption and authentication,
- database configuration to provide access control and auditing mechanisms,
- client security, and
- policies for users and administrators.

This chapter will concentrate on securing the database host, in order to protect against *back door attacks*. This does nothing to address abuses of data and/or systems by system administrators and legitimate users.

2. Who Has a Legitimate Need For Access?

There are two kinds of access: physical and logical. It is generally very desirable to provide physical security for the server if at all possible, because otherwise disks can be carried away to be read elsewhere, tape backups can be made, and boot procedures can often be over-ridden. Physical protection can be provided by locking the machine in a room where the only people who are authorized to enter are the database and system administrators. If it is placed in a large common machine room where multiple people can get to it, there will be risks from attacks that would not otherwise be possible.

The database may need to be logically accessed by the following persons (some of whom may perform multiple tasks):

- one or more UNIX system administrators (with root password)
- one or more database system administrators
- one or more database security administrators
- the database users

However, the form of access that is provided to the system should certainly vary. In particular, *there should be no need for the database users to have UNIX accounts on the server.* Moreover, it is a fairly large security hole to do so. If a user can gain access to a shell on most UNIX machines, there are generally holes that can be found in `setuid` system utilities that will allow them to gain root privileges. Examples of past compromises include the use of `rdist`, `sendmail`, `/bin/login`, and `/bin/mail`. Moreover, even if the legitimate user does not have this ability or knowledge, legitimate users logging into the server will create an opening to break into the machine, through a variety of mechanisms such as password sniffing or badly chosen or improperly handled passwords.

Access to the database itself need not go through the front door. In particular, if the UNIX system administrator is not the same as the database administrator, the UNIX system administrator can still read the data from the raw device file where the database is located using standard UNIX utilities such as `cpio`, `dump`, `dd`, or `cat`. The interfaces to these utilities are not particularly easy to use for the average user, but are relatively easy for a competent system administrator. Using these utilities, it would be simple to replicate the raw partition to a tape and move it to another database server.

In the event that there is more than one system administrator with knowledge of the root password, one should probably institute a policy that they log in under their own identity and execute an `su` command to obtain root privileges. The method for doing this is system-dependent (e.g., `/etc/ttytab` in SunOS), but it can usually be enforced to disallow remote root logins. This is useful to instill responsibility in administrators, but can also be used to keep track of who makes administrative changes.

3. Use a Firewall or Secure the Server?

Unfortunately, the definition of access as discussed so far is simplistic. Computers are often installed as part of a larger information infrastructure, and may in fact interact with processes on other computers on the network. For example, systems generally need to be backed up, and this is often done on a facility-wide system rather than on an individual system. Such network services can, however, be a gaping hole to someone trying to penetrate the database server, and such things need to be planned in such a way that they do not compromise security. For example, the BudTool backup system requires that the backup server perform an `rsh` with root privileges to systems being backed up, and this leaves a gaping hole in what might otherwise be a very secure system. Commercial products such as this should be examined to ensure that they do not open up holes.

In the current world of internetworked computers, one can usually identify sets of systems that need to trust each other, and sets of computers that need to communicate with each other without necessarily trusting each other. For example, the previously mentioned problem of backing up a network of computer systems generally requires the backup system to be trusted (unless a cryptographic mechanism is used to identify the backup system to the systems being backed up). It often makes sense to draw a fence around machines within a single administrative trust domain, and protect these machines from outside untrusted machines. Such a fence is called a firewall, and can greatly simplify the task of administering security by allowing attention to be focused on the boundary where trust is not universal. It is important to remember, however, that trust is transitive. That is, if machine A trusts machine B, and machine B trusts machine C, then machine A is implicitly trusting machine C. Boundary machines of a firewall (often called bastion hosts) therefore need to be configured very carefully to limit this transfer of trust.

In the example of the Oceania CRADA, the SQL server might be grouped together with machines such as an administrator's workstation, backup server, mail server, or Kerberos security server, but separated from database user workstations. The most obvious communication that needs to take place between the trusted Local Area Network (LAN) and user workstations is SQL. Sybase uses a very simple communication between clients and servers that is well-suited to protection by a firewall, namely communication over a single TCP port connection to the server, on a destination port that is chosen at installation time. Firewall architecture is beyond the scope of this discussion, but this kind of communication can easily be handled with a packet filter provided by a router or dual-hosted firewall. If this is the only service required from the server by the outside world, then a boundary router with an access control list allowing only the SQL port to pass inside will be a very simple solution. For more information on firewalls, consult the excellent books by Cheswick and Bellovin [Cheswick 1994] and Chapman and Zwicky [Chapman 1995].

In the event that further services are required to be passed from the trusted database server to untrusted machines, a full-blown and potentially complicated firewall will likely be needed. For example, SMTP mail service must be configured carefully to protect against known security weaknesses in `sendmail`. There are times when existing network architectures or performance considerations make it impractical to use a firewall. For example, firewalls are usually dependent on packet filters provided by routers or UNIX machines that function as routers. In a network architecture such as switched ethernet or ATM, this may not be possible. In this case, much more effort needs to be applied to securing the database server.

It is now commonplace to provide remote network access via dial-up terminal servers that run SLIP or PPP. One of the biggest problems with using this kind of network access is that most PC dialup software does not allow for support of one-time passwords for strong authentication. In particular, this author is not aware of any configurations to support the OSF Distributed Computing Environment (DCE). As a result, dialup terminal servers can become a target of attack, and can be very difficult to secure (or audit).

This raises the question about *where* to place the firewall, and in particular whether to assume that the database clients are within the trusted domain secured by a firewall. There is no simple answer to this question, since it will depend on the function of the clients, the other systems they interact with, and the level of control that can be placed on clients. In the case of dial-up clients, however, one should probably assume that they are *outside* a firewall, and provide only the access that they need. This probably means only allowing SQL and DCE authentication requests to come through the firewall.

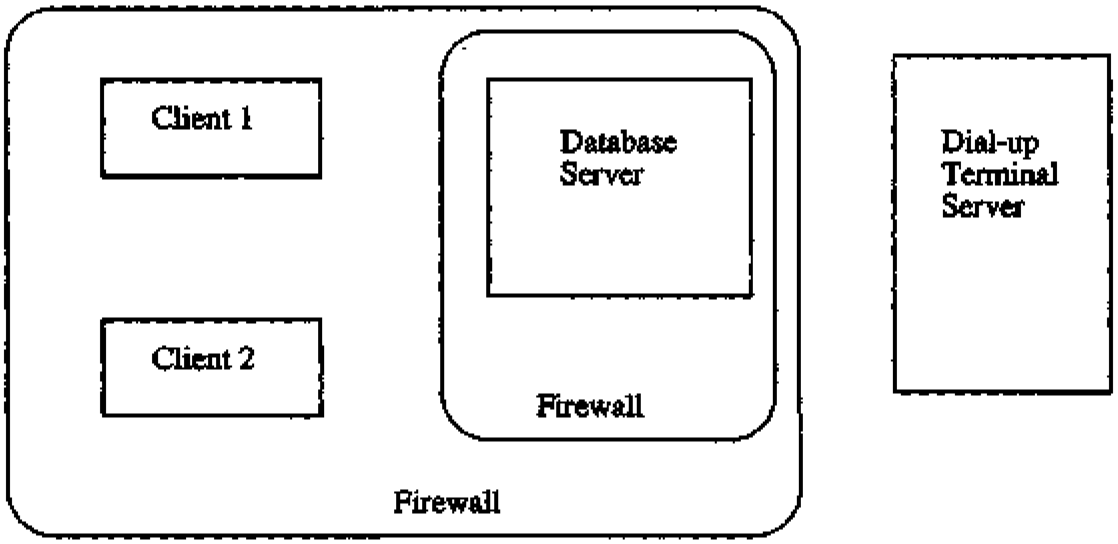


Figure 3.1 Firewalls

Firewalls can be placed around the database server, around a domain containing the server and the clients, or in both locations. The purpose of a firewall is to limit the interaction between two domains, and they can be configured to protect servers from clients, and to protect clients from the outside world.

4. Tools for Securing Machines

There are a number of tools and products that can be used to secure a UNIX system. Among the more notable are:

SATAN	This tool is available from the Internet at various sites. It is not a marketed product and can be tricky to use and install. It's quite useful however if one has the time.
ISS	The Internet Security Scanner is used for probing machines from outside to reveal weaknesses.
tcp_wrappers	This set of tools can be used to limit the IP addresses that connect to a given service started from `inetd`. It is very useful, and available for free on the Internet [Purdue].
Kerberos	More on this later.
DCE	See the appendix discussing DCE.
S/Key	S/Key is a challenge-response user authentication system that can be used to replace passwords. This has recently been superseded by OPIE, and is available from the Internet [Navy].

TIS firewall toolkit	This is an early freeware version of the Gauntlet firewall product. It is not a complete firewall, but a useful set of tools for building a firewall. For a serious installation, you should probably purchase the commercial version.
Other firewalls	The last few years have brought an explosion in the number of vendors selling firewall products. There has been very little reputable evaluation work done on these products, and most of them are designed for different purposes than the requirements of the Oceania product. Some vendors that appear to be quite reputable are TIS, V-One, and Secure Computing Corporation. A large list of firewall products and vendors is provided on the Internet [GreatCircle].
Security Dynamics SecurID	A handheld authentication replacement for passwords. The kit that they sell includes software for the host that replaces the login shell with a proprietary program to call the authentication server. In comparison to other handheld devices, this one has some advantages in the battery life and relatively little typing required by the user.
Tripwire	Once you think you have secured the database server, you then have the problem of maintaining your confidence in it. One tool that can be very useful for this is Tripwire, which keeps a database of cryptographic checksums of critical files and programs that will allow you to detect changes caused by hackers. It is available from the COAST project at `ftp://coast.cs.purdue.edu/pub/tools/unix/Tripwire`.

A very useful list of security tools is available online [CIAC A, 1996].

5. Network Authentication

Given that the number of accounts on the server should be severely limited, there are still situations in which remote logins to the machine are required, for example to perform system backups, database administration, or other system maintenance tasks. This is particularly true if the machine is located in a physically secure location such as a locked computer room. In such cases, it is extremely desirable to use a mechanism whereby the remote login session is strongly authenticated with cryptography, and the contents of the communication are also encrypted to prevent eavesdropping. The Kerberos network security system can provide both services, and is strongly recommended if remote maintenance is required. Without it, attacks can be mounted against the system using password sniffing, TCP session takeovers, or password guessing. This is particularly important if the network used to connect the server and administrator workstation has untrusted users or machines located on it.

5.1 Which Kerberos?

Kerberos was originally developed under U.S. government funding at M.I.T., and version IV gained wide acceptance among a variety of sites. There were however weaknesses and deficiencies in version IV, so multiple organizations undertook to enhance it. For example, Sandia National Laboratories undertook to write extensions for using handheld authentication tokens, but these changes were never folded into the M.I.T. distribution directly. The most notable version of Kerberos now available is M.I.T.'s version 5, which DCE Security Services are based on. The former is available in source code form, but comes with no support from M.I.T.. Third party support is available from several sources however, including Cygnus Software in Mountain View, California [Cygnus, 1996]. DCE Security Services offer extensions including access control lists and groups, and DCE Security Services are also used in DCE's Distributed File Service (DFS). DCE is generally regarded as more powerful, but also includes more complexity, more expense, and only limited support of server platforms. Information on DCE is available from OSF [OSF/DCE, 1996].

One of the strongest factors in favor of using Kerberos is that it provides a key management infrastructure to support "single login", where a user has a single method of logging into all machines that are within a single administrative domain. In particular, in theory this can be used to secure access to the database, logins on the database server via telnet, logins on the clients, and logins on the terminal server. In practice, support for Kerberos is not universal. It should be noted that all UNIX machines support Kerberos, and Xyplex terminal servers support it as well. In addition, Microsoft has announced support for Kerberos/DCE in the next version of NT (Cairo) [MicroSoft A, 1996] [MicroSoft B, 1996].

6. UNIX *Can* Be Made Secure

UNIX machines have been the perennial whipping boys of the computer security community. The reasons for this are twofold: (1) UNIX machines were primarily designed to provide a robust and rich set of services, but were designed to be open, not necessarily secure, and (2) UNIX machines are generally shipped with a default configuration that favors a multitude of services rather than security. For example:

- Sun systems default to have a plus sign in the `/etc/hosts.equiv` file, allowing root users from other machines to login as root without a password,
- SGI machines are shipped by default with several guest accounts that have no passwords.
- SGI machines are shipped with a default on their X windows display manager (xdm) configuration set so that all keystrokes are world readable.

Each of these problems can be easily corrected, but one must *know* to do so. It should be pointed out that SGI and Sun are not necessarily worse than the other vendors in security, but these are the ones that the authors are most familiar with. In spite of the fact that UNIX has a bad reputation in security, it is a fact that the overwhelming majority of Internet firewalls are built from specially configured UNIX machines, and a properly configured UNIX machine can provide both a high degree of functionality as well as a high degree of security in the face of capable adversaries. The devil is in the details however.

6.1 Turn Off Everything You Don't Need

The function identified for the server in the Oceania CRADA is very simple: provide database services to standard SQL commands. Anything beyond this function should be considered a potential risk. The purpose of a firewall is to mediate the communication between different trust domains so that only safe communication is allowed. The discussion of this section can be used to design such a firewall or to secure the database server in lieu of a firewall.

The previous section mentioned that the number of UNIX user accounts on the server machine should be severely limited. This generally precludes using such a server in a dual role as an electronic mail server (say using the POP protocol). For a large installation, this is completely reasonable because there are often other machines available for these other functions. For a small organization that wants to leverage a single machine for multiple functions, the situation becomes much more complicated. In particular, if the SQL server is also a POP server, then user accounts and passwords are required on the SQL server. If the SQL server is used as a World Wide Web server, then the machine is vulnerable to attacks through the `http` protocol. It's best to keep things simple.

Given that you have identified the services that you need to supply, the next thing to do is to turn off all other services. This is a somewhat haphazard procedure that depends on the particular brand of UNIX that is being used. In general network services are either started at boot time or else are started through the Internet services daemon `inetd`. A complete discussion of services is beyond the scope of this paper, but we discuss the most common ones here. Further information can be found in the books by Chapman and Zwicky [Chapman, 1995] or Garfinkel and Spafford [Garfinkel, 1996].

6.2 Services Started at Boot Time

Services started at boot time are usually accessed from the `/etc/rc*` hierarchy, although variations exist on different UNIX flavors. The following services need to be examined:

named	This is the domain name service. This is often necessary for convenience, but *in no case should security be based on it.* Whenever machines are referred to in access control lists (say on a router), the reference should be made by IP address only and not DNS names. This is necessitated by the fact that DNS uses no authentication, and is subject to spoofing. Note that there has recently been a dramatic increase in attacks using DNS corruption on the Internet.
sendmail/smtp	This is the protocol used for transferring mail. If at all possible, it should be disabled on the database host. This has been the single biggest cause of security compromises in UNIX machines.
NFS	NFS stands for Network File System. Most implementations are based on version 2.0, and are inherently insecure because the only authentication that is done is performed at initial mount time, and never after this. It is exceptionally convenient for sharing files, but in no case should it be used to share files that are critical to security. In particular, if user's home directories are exported or imported via NFS, then the `.rhosts` files are vulnerable to attack. If the `/etc` partition is exported from the server, then it will likely expose weaknesses on the server.
statd	This is used to provide remote `stat()` service for the Network File System. There is currently a weakness that has been discovered in UNIX fla-

vors involving the `statd` daemon, and is being exploited to break into machines. No patch is yet available for some vendors, so you should turn it off unless you absolutely need NFS.

syslog `syslog` is used for receiving and sending log messages. Unfortunately, bugs have been found in the `syslog()` routine of UNIX versions derived from BSD (it uses `gets()`), and this can be exploited to break into machines if it has not been patched. On the other hand, logging things can be very useful to detect break-in attempts and service failures. Make sure you pay attention to how it is configured, and try to make sure that the server does not accept syslog information from just any machines. It may also be useful to do remote logging of security and service events such as failed logins and TCP/IP portscans. This can be done for clients, network routers, and terminal servers.

rpcbind and portmapper These are used for RPCs (remote procedure calls) to support other services. Unfortunately, network services that go through most vendor-supplied portmapper programs cannot be trusted, and should be disabled if possible. In any event you should be careful to screen the portmapper and RPC services against untrusted machines.

6.3 Services Typically Started in `/etc/inetd.conf`

At boot time, a master daemon is started called `inetd` (Internet Services Daemon). The purpose of this process is to listen to the network and answer calls to start other programs to provide services. The configuration of `inetd` is contained in the file `/etc/inetd.conf`. Consult your UNIX vendor documentation for the exact syntax.

ftp This is used to transfer files across networks. It is often required, but should be configured to use strong authentication such as Kerberos, S/Key, or a hand-held authenticator. **TURN OFF ANONYMOUS FTP.**

telnet This is used for remote login. It should be configured to use strong authentication via one-time passwords. This can be done with Kerberos, S/key, or a variety of other products. Beware also that some telnet server programs have been discovered to have bugs related to the passing of environment variables. Make sure that your version has the latest security patches from the vendor.

shell/login/exec This is the Berkeley suite of remote operations. They should be disabled as they are inherently insecure. They rely on passwords or `.rhosts` files placed in the home directories of users.

talk This is a remote talk protocol for interactively typed conversations. Turn it off.

finger This is useful for finding out who is actively logged onto the server, but it can also be used to reveal usernames for password guessing attacks. Weaknesses in older versions of the `finger` daemon were used in the Morris Internet worm. Turn it off.

tftp	This is sometimes used for booting X terminals, terminal servers, routers, or diskless workstations. It has no authentication whatsoever, and if it is improperly configured can be used to fetch absolutely any file from the server. Other devices on the network should be configured to boot from local flash cards or other `tftp` servers. Turn it off.
systat	Turn it off. It allows others to invoke a `setuid` executable to see what processes are being run.
netstat	Turn it off. It can be used to see what network connections are open to remote machines.
uucp	This is a crufty old mechanism used to transfer mail and files, primarily over dialup lines. Turn it off on the server.
time	This is an old protocol for keeping clocks of machines on a network synchronized. A much better alternative is NTP, the Network Time Protocol. Source for NTP is available for free on the Internet. Note that a reliable time source is needed for timestamping, so you should pay particular attention to this if you intend to rely on the machine's notion of time.
mountd	This is part of NFS (Network File System). It is generally a very bad idea to export file systems from the server, and in any event should only be done in a read-only mode. If NFS is not required, then turn it off.
rexd	This is an insecure remote execution protocol. Turn it off.
rquotad	This is used to support quotas for NFS clients. Turn it off.
rusersd	This is used to determine what users are logged into the server, and should be turned off just like `finger`.
sprayd	This service is used for network testing, but is generally redundant with other services. Turn it off.
walld	This is used for broadcasts to all users logged into a machine. Turn it off.

The advice given here is not absolute. You might actually need some of these services for your network environment. In the event that you need to leave one or more running, you will need to evaluate options for securing them or screening them from untrusted machines.

6.4 Other services

Once you have gone through the list of services described here, you should reboot your system and go over it again. First run `ps` to see what processes are still running. The list should be very small, and you should be able to identify what each and every process is there for. Some of these (like `init`) are part of the kernel and need to be there. Others should be tracked down to determine if they are needed or if they pose a risk.

One method you can use to identify dangerous processes is to run `netstat` to see what network connections are available or active. As an alternative, you can install a port scanner to see what services are still running. There are several commercial product (ISS is a reputable product), but you can also use the freely distributed scanners from the SATAN network security scanner, or the TIS firewall toolkit. Pointers to the source code for these can be found through the CERT [SEI, 1996] and CIAC World Wide Web pages [CIAC B, 1996]. CERT stands for Computer Emergency Response Team, and CIAC stands for Computer Incident Advisory Capability. The former serves the university research community, and the latter serves the Department of Energy complex. Both offer an excellent source of information on securing internetworked systems.

Chapter 4 - Database Specifics

This chapter will discuss the following specific database issues:

- Database User Authentication,
- Database User Authorization, and
- Database Event Auditing.

1. Database User Authentication

Oceania's current implementation utilizes Sybase's supplied authentication service. Access is controlled through a SQL Server login and password (not an operating system login and password).

1.1 Sybase Logins

When a login is established, the user can connect to the server and use any database to which he or she has permission.

1.1.1 How Logins Work

When a new login is created, a row is added to a system table. Each login is assigned a system identifier that identifies that user uniquely in the server. During login, the server matches the name passed in the login structure against the name column in a system table. If a match is found and the password matches, the user's identifier is stored in the memory allocated for the new connection.

The login name is not stored in any table except the system table. The user's identifier is the key used in all other tables that relate to a person's login, including system tables assigning roles and those relating server access to database access.

1.1.2 System Administrators

System Administrators are special users who handle tasks not specific to applications. SQL Server recognizes a System Administrator as a super-user who works outside SQL Server's command and object protection system. The role of System Administrator is usually granted to individual SQL Server logins. This provides a high degree of individual accountability because all actions taken by that user can be traced to his or her individual server user ID.

The fact that a System Administrator operates outside the protection system serves as a safety precaution. For example, if the Database Owner accidentally deletes all the entries in the `sysusers` table, the System Administrator can recover the information (provided, of course, that the system is being backed up regularly).

When Release 10.0 of SQL Server is installed, it still has the default "sa" account, which has System Administrator, System Security Officer, and Operator roles enabled. For greater accountability for the highly privileged users in your system, it is recommended that you create individual login accounts for users who are to be granted these privileges, grant them their roles, and then lock the "sa" account.

1.2 Sybase Passwords

In Sybase SQL Servers of release 10.0 and later (currently used by Oceania), passwords may be encrypted on the client side before passing across a network. If password encryption is desired, the following occurs:

- The initial login packet is sent without passwords.
- The client indicates to the remote server that encryption is desired.
- The remote server sends back an encryption key, which the client uses to encrypt its plaintext passwords.
- The client then encrypts its own passwords, and the remote server uses the key to authenticate them when they arrive.

This scheme is vulnerable to the following attacks:

- Anyone who overhears or intercepts the key in transit can later read all passwords encrypted using that key.
- A man-in-the-middle attack, where someone masquerades as the server, intercepts the key, passes it to the user, and then authenticates to the server with the response.
- A password guessing attack.

2. Database User Authorization

Once clients (i.e., users) are authenticated, the server application is responsible for verifying which operations are permitted on the information being accessed. A requirement placed on the system by one of Oceania's customers is that a "user's rights are identical whether the interaction is through the EMR (Electronic Medical Record) application or third party SQL tools"[1]. It will be necessary to implement a sophisticated privilege scheme in order to satisfy this requirement.

This section will briefly discuss DBMS object privileges and present several different table privilege granting schemes. It assumes a basic understanding of Sybase authorization mechanisms. The examples presented will graduate from one that will provide no security at all to one that provides row-level security enforced within the database server. The section will not discuss field-level security in any detail, but notes that the examples could be extended to provide for such granularity.

2.1 Object Privileges

To obtain access to database tables, views, and procedures, users must either have privileges based on role, special user status, or group membership, or be granted explicit permission for each type of access. Database and object owners can grant or revoke permission on objects they own. For each permission granted, the grantor can specify that the recipient can grant the permission to another user. This form of delegating permission is called "granting with grant". When revoking permission, the revoker can specify that the permission be revoked from all users to whom the recipient granted it.

You can grant or revoke the following object permissions:

- `SELECT` select data from a table or view

1. Oceania Inc., *Security Functional Requirements V2.2*, January 30, 1996

• `INSERT`	insert a row in a table or view
• `DELETE`	delete a row in a table or view
• `UPDATE`	update a row in a table or view
• `EXECUTE`	execute a procedure

Providing and enforcing user authorizations to the table-level within a relational database is quite trivial. Permissions to tables may be granted and revoked to and from users and groups of users and enforced within the database server. Providing row-level security, however, is another matter. Providing and enforcing user authorizations to the row-level or field-level can be accomplished within the client application, but is useless if a user accesses the database through a third-party query tool or one of the database vendor supplied utilities.

2.2 Table Definitions

In order to illustrate some privilege schemes, the following table definitions including subsets of fields from the respective Oceania tables will be used:

• LdbAccess	defines which application users can access which patient records
• LdbPatient	holds information about patients
• LdbProvider	holds information about providers
• LdbPPatientList	assigns patients to providers

Table 1: LdbAccess

Field Name	*Data Type*	*Null*	*Primary Key*
LdbPatientID	char(10)	not allowed	X
LdbUserID	char(10)	not allowed	X

Table 2: LdbPatient

Field Name	*Data Type*	*Null*	*Primary Key*
LdbPatientID	char(10)	not allowed	X
SSN	varchar(11)		
LastName	varchar(32)		
FirstName	varchar(32)		
MiddleName	varchar(32)		
DateOfBirth	datetime		

Table 2: LdbPatient

Field Name	*Data Type*	*Null*	*Primary Key*
PrimaryPhysician	char(10)		

Table 3: LdbProvider

Field Name	*Data Type*	*Null*	*Primary Key*
LdbProviderID	char(10)	not allowed	X
SSN	varchar(11)		
LastName	varchar(32)		
FirstName	varchar(32)		
MiddleName	varchar(32)		
UserID	int		

Table 4: LdbPPatientList

Field Name	*Data Type*	*Null*	*Primary Key*
LdbProviderID	char(10)	not allowed	X
LdbPatientID	char(10)	not allowed	X

2.3 All Privileges

The simplest implementation of table access grants `SELECT`, `INSERT`, `UPDATE`, and `DELETE` on every database table to the `public` group (see the following table). Every user defined by the database is automatically a member of the `public` group and may then perform queries against all tables granted to the group. Where all privileges are granted to the `public` group, any user may insert new rows, and select, update and delete existing rows within the tables. Essentially, no security is enforced.

Table 5: All Privileges

Object	*Object Type*	*Permission*	*Subject (Group)*

LdbAccess	User Table	`SELECT`
		`INSERT`
		`UPDATE`
		`DELETE`
			public

LdbPatient	User Table	`SELECT`
		`INSERT`
		`UPDATE`
		`DELETE`
			public

LdbPPatientList	User Table	`SELECT`
		`INSERT`
		`UPDATE`
		`DELETE`
			public

LdbProvider	User Table	`SELECT`
		`INSERT`
		`UPDATE`
		`DELETE`
			public

2.4 Granting Specific Table Privileges to Users and Groups

To limit query activity to the table level, all privileges could be revoked from the `public` group and specific privileges granted to application specific groups (and/or users). For instance, `SELECT` could be granted on a particular table for some groups of users and `INSERT`, `UPDATE`, and `DELETE` be explicitly revoked on the table for other groups of users. In this case, a user who is the member of a group possessing the `SELECT` privilege may select (i.e., retrieve) all rows in the table. However, he or she may not insert new rows or update or delete existing rows without the appropriate privileges. This scheme provides no row-level security. The privileges granted allow queries to be performed on all rows within the respective tables.

The following groups of users will be used for the examples:

• OC_SYSADMIN	System Administrators,
• OC_PHYS	Physicians, and
• OC_RN	Registered Nurses.

The following table reflects such a scheme. It is not, necessarily, intended to duplicate Oceania's use of group privileges.

Table 6: Specific Table Privileges

Object	*Object Type*	*Permission*	*Subject (Group)*

LdbAccess	User Table	`SELECT`
		`INSERT`
		`UPDATE`
		`DELETE`
			OC_SYSADMIN

LdbPatient	User Table	`SELECT` `UPDATE`	OC_SYSADMIN OC_PHYS OC_RN
LdbPatient	User Table	`INSERT` `DELETE`	OC_SYSADMIN
LdbPPatientList	User Table	`SELECT`	OC_SYSADMIN OC_PHYS OC_RN

LdbPPatientList	User Table	`INSERT`
		`UPDATE`
		`DELETE`
			OC_SYSADMIN

LdbProvider	User Table	`SELECT`	OC_SYSADMIN OC_PHYS OC_RN
LdbProvider	User Table	`UPDATE`	OC_SYSADMIN OC_PHYS
LdbProvider	User Table	`INSERT` `DELETE`	OC_SYSADMIN

Referring to the LdbPatient table in explaining the above table of privileges, notice that users who are members of the OC_SYSADMIN, OC_PHYS, and OC_RN groups have the ability to select (and view) and update existing rows in the table. Only members of the OC_SYSADMIN group have the ability to insert new rows (i.e., patients) and delete existing rows.

Privileges as granted in this case are granted to users whether they are accessing the database through the client application (i.e., WAVE), a third-party query tool, or database server utilities like `ISQL`. They do not, however, provide row-level security.

2.5 Basic Stored Procedures

For this discussion, a basic stored procedure is one that performs an elementary `SELECT`, `INSERT`, `UPDATE`, or `DELETE` transaction against an individual table. Extending the previous example, some row-level security may be enforced by performing all database updates with basic stored procedures.

These basic stored procedures must be called from the user interface or some functional layer between the user interface and the database server. They require input parameters and return result sets and/or status information.

Here are some example basic stored procedures for the LdbPatient table assuming the following naming conventions:

- dlet*tablename* for all DELETE procedures,
- nsrt*tablename* for all INSERT procedures, and
- upd*tablename* for all UPDATE procedures.

2.5.1 INSERT Stored Procedure

```
create procedure nsrtLdbPatient
     (@LdbPatientID char(10) = null,
     @SSN varchar(11) = null,
     @LastName varchar(32) = null,
     @FirstName varchar(32) = null,
     @MiddleName varchar(32) = null,
     @DateOfBirth datetime,
     @PrimaryPhysician char(10) = null)
as
-- Procedure for inserting a row in the LdbPatient ta-
ble
if @LdbPatientID = null
     begin
           print "Invalid Patient identifier"
           return -100
     end
INSERT into LdbPatient
values (@LdbPatientID,
     @SSN,
     @LastName,
     @FirstName,
     @MiddleName,
     @DateOfBirth,
     @PrimaryPhysician)
if @@transtate = 2
```

```
          begin
                rollback tran
                return
          end
     commit tran
     return
```

2.5.2 UPDATE Stored Procedure

```
     create procedure updtLdbPatient
          (@LdbPatientID char(10) = null,
          @SSN varchar(11) = null,
          @LastName varchar(32) = null,
          @FirstName varchar(32) = null,
          @MiddleName varchar(32) = null,
          @DateOfBirth datetime,
          @PrimaryPhysician char(10) = null)
     as
     -- Procedure for updating a row in the LdbPatient ta-
     ble
     if @LdbPatientID = null
          begin
                print "Invalid Patient identifier"
                return -100
          end
     UPDATE LdbPatient
     set SSN = @SSN,
          LastName = @LastName,
          FirstName = @FirstName,
          MiddleName = @MiddleName,
          DateOfBirth = @DateOfBirth,
          PrimaryPhysician = @PrimaryPhysician
     where LdbPatientID = @LdbPatientID
     if @@transtate = 2
          begin
                rollback tran
```

```
              return
         end
    commit tran
    return
```

2.5.3 DELETE Stored Procedure

```
create procedure dletLdbPatient
(@LdbPatientID char(10) = null)
as
-- Procedure for deleting a row in the LdbPatient ta-
ble
if @LdbPatientID = null
begin
print "Invalid Patient identifier"
return -100
end
DELETE LdbPatient
where LdbPatientID = @LdbPatientID
if @@transtate = 2
begin
rollback tran
return
end
commit tran
return
```

2.5.4 Basic Stored Procedures Privileges

In order to duplicate the functionality described in the section, Specific Table Privileges, privileges to execute the stored procedures would be as follows:

Table 7: Basic Procedure Privileges

Object	*Object Type*	*Permission*	*Subject*
LdbAccess	User Table	SELECT	OC_SYSADMIN

Table 7: Basic Procedure Privileges

Object	*Object Type*	*Permission*	*Subject*
nsrtLdbAccess	Procedure	EXECUTE	OC_SYSADMIN
updtLdbAccess	Procedure	EXECUTE	OC_SYSADMIN
dletLdbAccess	Procedure	EXECUTE	OC_SYSADMIN
LdbPatient	User Table	SELECT	OC_SYSADMIN OC_PHYS OC_RN
nsrtLdbPatient	Procedure	EXECUTE	OC_SYSADMIN
updtLdbPatient	Procedure	EXECUTE	OC_SYSADMIN OC_PHYS OC_RN
dletLdbPatient	Procedure	EXECUTE	OC_SYSADMIN
LdbPPatientList	User Table	SELECT	OC_SYSADMIN OC_PHYS OC_RN
nsrtLdbPPatientList	Procedure	EXECUTE	OC_SYSADMIN
updtLdbPPatientList	Procedure	EXECUTE	OC_SYSADMIN
dletLdbPPatientList	Procedure	EXECUTE	OC_SYSADMIN
LdbProvider	User Table	SELECT	OC_SYSADMIN OC_PHYS OC_RN
nsrtLdbProvider	Procedure	EXECUTE	OC_SYSADMIN
updtLdbProvider	Procedure	EXECUTE	OC_SYSADMIN OC_PHYS
dletLdbProvider	Procedure	EXECUTE	OC_SYSADMIN

Although the above example procedures and privileges provide no more security than granting specific privileges on tables to groups, there are a couple of immediate advantages:

- SQL to perform the inserts, updates, and deletes has been removed from the client application.
- The procedure source code may be programmatically generated given the table definitions stored in the database server.

The next section will extend the procedures in order to illustrate how further levels of security could be accomplished.

2.6 Extended Basic Stored Procedures

In both, the Specific Table Privileges examples and the Basic Stored Procedures examples, we noticed that any member of the OC_PHYS group can update any and all existing rows in the LdbPatient table. To extend the examples without creating new database tables or modifying the existing ones, we will make the following assumptions:

- The values stored in the LdbUserID field in the LdbAccess table match user names in the database server.
- The LdbAccess table identifies which LdbPatient occurrences can be updated by the current user.

The examples presented will extend the `UPDATE` and `DELETE` stored procedures for the LdbPatient table. Permissions to execute the procedures will remain as above. Inserts to the table will be discussed in a later section.

2.6.1 Extended UPDATE Stored Procedure

```
create procedure updtLdbPatient
(@LdbPatientID char(10) = null,
@SSN varchar(11) = null,
@LastName varchar(32) = null,
@FirstName varchar(32) = null,
@MiddleName varchar(32) = null,
@DateOfBirth datetime,
@PrimaryPhysician char(10) = null)
as
-- Procedure for updating a row in the LdbPatient ta-
ble
if @LdbPatientID = null
begin
print "Invalid Patient identifier"
return -100
end
-- Begin extension to provide row-level security
-- Check to see if the current user has been provid-
ed access to the patient
-- Accepting the current user's name from the server
assures that the user
-- has successfully completed the authentication proc-
ess.
if not exists (SELECT * from LdbAccess
where LdbPatientId = @LdbPatientID
and LdbUserID = suser_name()
begin
print "Access to patient denied."
return -200
end
-- End extension to provide row-level security
UPDATE LdbPatient
set SSN = @SSN,
LastName = @LastName,
FirstName = @FirstName,
MiddleName = @MiddleName,
DateOfBirth = @DateOfBirth,
PrimaryPhysician = @PrimaryPhysician
where LdbPatientID = @LdbPatientID
if @@transtate = 2
begin
rollback tran
return
end
commit tran
return
```

2.6.2 Extended DELETE Stored Procedure

(just like the UPDATE procedure)

```
create procedure dletLdbPatient
(@LdbPatientID char(10) = null)
as
-- Procedure for deleting a row in the LdbPatient ta-
ble
if @LdbPatientID = null
begin
print "Invalid Patient identifier"
return -100
end
```

```
-- Begin extension to provide row-level security
-- Check to see if the current user has been provid-
ed access to the patient
-- Accepting the current user's name from the server
assures that the user
-- has successfully completed the authentication proc-
ess.
if not exists (SELECT * from LdbAccess
where LdbPatientId = @LdbPatientID
and LdbUserID = suser_name()
begin
print "Access to patient denied."
return -200
end
-- End extension to provide row-level security
DELETE LdbPatient
where LdbPatientID = @LdbPatientID
if @@transtate = 2
begin
rollback tran
return
end
commit tran
return
```

2.6.3 Extended INSERT Stored Procedure

In discussing INSERT privileges, we will continue using the LdbPatient table as our example. Since the inserting of an LdbPatient row establishes a new patient with a new identifier (i.e., LdbPatientID), we know that a row with the same patient identifier cannot exist in the LdbAccess table prior to the insert. We note from the privileges table above that members of the OC_SYSADMIN group are the only users capable of inserting rows into the LdbPatient table. And, members of the OC_SYSADMIN group are the only users that can insert into the LdbAccess table. Therefore, only members of the OC_SYSADMIN group can insert rows into the LdbPatient table and the LdbAccess table. This is important because one would assume that a user who is authorized to insert new patient information should also be able to update or delete the information (i.e., grant herself or himself permissions to do so).

The following INSERT procedure establishes the access permissions for the user entering new patient information. It does not establish privileges for any other users.

```
create procedure nsrtLdbPatient
(@LdbPatientID char(10) = null,
@SSN varchar(11) = null,
@LastName varchar(32) = null,
@FirstName varchar(32) = null,
@MiddleName varchar(32) = null,
@DateOfBirth datetime,
@PrimaryPhysician char(10) = null)
as
-- Procedure for inserting a row in the LdbPatient ta-
ble
if @LdbPatientID = null
begin
print "Invalid Patient identifier"
return -100
end
INSERT into LdbPatient
values (@LdbPatientID,
@SSN,
@LastName,
@FirstName,
@MiddleName,
@DateOfBirth,
@PrimaryPhysician)
if @@transtate = 2
begin
rollback tran
return
end
-- Begin extension to provide row-level security
-- Insert privileges to patient for current user
INSERT into LdbAccess
values (@LdbPatientID,
suser_name())
if @@transtate = 2
```

```
begin
rollback tran
return
end
-- End extension to provide row-level security
commit tran
return
```

2.6.4 Extended Stored Procedures Summary

This section illustrated one method for providing and enforcing privileges to insert, update, and delete rows in and from the LdbPatient table to the current user. The examples utilized existing Oceania table definitions without additional tables or attributes. User/group privileges to tables and procedures were modified to accommodate the examples. There were some necessary assumptions made during the process. The procedure code was intended to provide easily understood examples and is not necessarily performance tuned.

The section does not address the issue of row-level privilege management. The `INSERT` procedure illustrated how the user inserting the new patient information could be automatically granted access to update and delete the respective patient information. It does not, however, address privileges to be granted to other users. It also does not address the issue of revoking privileges. However, the `DELETE` procedure does remove privileges from the LdbAccess table when a patient is removed from the LdbPatient table. Furthermore, none of the procedures address referential integrity issues that may need to be programmatically maintained when considering dependent relationships.

2.7 Using Views

Enforcing row-level security during retrievals can be enforced through database procedures or views. This section will present some examples of using views. As before, there will be no additional attributes or tables, user/group privileges to tables will remain the same as in the previous section, and direct access to tables is prohibited.

We will begin with the view definitions required.

Table 8: View Definitions

View Name *[comment]*	*Definition*
Access [select all rows and all columns from the LdbAccess table that meet the supplied where clause criteria]	`create view Access as SELECT * from LdbAccess`

Table 8: View Definitions

View Name *[comment]*	*Definition*
Patient **[select all rows and all columns from the LdbPatient table that meet the supplied** `where` **clause criteria and where the user has been granted access to the patient in the LdbAccess table]**	`create view Patient as SELECT b.* from LdbAccess a, Ldb-Patient b where a.LdbUserID = suser_name() and b.LdbPatientID = a.LdbPatien-tID`
PPatientList **[select all rows and all columns from the LdbPPatientList table that meet the sup-plied** `where` **clause criteria and where the user has been granted access to the patient in the LdbAccess table]**	`create view PPatientList as SELECT b.* from LdbAccess a, Ldb-PPatientList b where a.LdbUserID = suser_name() and b.LdbPatientID = a.LdbPatien-tID`
Provider **[select all rows and all columns from the LdbProvider table that meet the supplied** `where` **clause criteria]**	`create view Provider as SELECT * from LdbProvider`

Table 9: View Privileges

Object	*Object Type*	*Permission*	*Subject*
LdbAccess	User Table	none	n/a
Access	View	`SELECT`	OC_SYSADMIN
LdbPatient	User Table	none	n/a
Patient	View	`SELECT`	OC_SYSADMIN OC_PHYS OC_RN
LdbPPatientList	User Table	none	n/a
PPatientList	View	`SELECT`	OC_SYSADMIN OC_PHYS OC_RN
LdbProvider	User Table	none	n/a

Table 9: View Privileges

Object	*Object Type*	*Permission*	*Subject*
Provider	View	SELECT	OC_SYSADMIN OC_PHYS

Notice in the View Privileges table that SELECT permissions are the only permissions granted to the views. It is possible to insert, update, and delete rows from views. It is, also, possible to enforce certain update rules with views. There are, however, caveats. For instance:[1]

- DELETE statements are not allowed on multi-table views.
- INSERT statements are not allowed unless all non null columns in the underlying table or view are included in the view through which you are inserting new rows.
- You cannot insert a row through a view that includes a computed column.
- INSERT statements are not allowed on join views created with `distinct` or `with check option`.
- UPDATE statements are allowed on join views `with check option`. The update fails if any of the affected columns appears in the `where` clause, in an expression that includes columns from more than one table.
- If you insert or update a row through a join view, all affected columns must belong to the same base table.
- You cannot update or insert into a view defined with the `distinct` clause.
- Data update statements cannot change any column in a view that is a computation, and cannot change a view that includes aggregates.

The limits described above prohibit us from duplicating all of the functionality provided by the stored procedures in the previous sections with views alone. We could, however, enforce retrieval (i.e., SELECT) row-level security through views and UPDATE and DELETE row-level security through stored procedures. Providing and enforcing row-level security when inserting rows, becomes another issue.

It is easy to grant permissions to select, update, and delete particular rows to the user who inserted the rows. It would also be easy to grant permissions to select, update, and delete those rows to users that are members of the same group as the user who inserted them. It would just be a matter of keeping track of users and the group or groups to which they belong. How would we, though, grant permissions to a particular user or group of users to insert a particular row before we know about the row? Let's use an example. Say, Bob is a legitimate user of the system and a member of the OC_SYSADMIN group with permission to insert rows into the LdbPatient table. While using the system, Bob has the need to enter information about a new patient, John. Since Bob has permission to insert into the table, should the system let him insert information concerning John, or should the insert be authorized by John or some representative of John? If John was required to authorize the insert, how would he interact with the system? If John was unable to interact with the system and there were no representatives available, how would the insert be authorized? How could the system grant permission

1. Sybase SQL Server v10.0 Reference Manual Volume 1, 1994

to Bob to insert information concerning John if it doesn't even know about John? This paper will not attempt to answer these questions, only to point out the simple example. The paper does assume that if Bob has the privileges necessary to insert a row into the LdbPatient table, then he possesses the necessary authorization to insert information about John.

2.8 Labeling of Subjects and Objects

Secure Relational Database Management Systems (SRDBMS) as provided by some vendors implement sensitivity labeling of subjects and objects in order to provide trusted transactions and multi-level security. These SRDBMSs, however, are probably not practical for most non-military applications. For instance, Sybase's Secure SQL Server requires a secure server operating system and expensive client software. It also labels such objects as databases and tables and provides certification of stored procedures that may be considered unnecessary for most applications.

It would be desirable, however, to implement some of the functionality provided by secure RDBMSs in a non-secure database application, especially sensitivity labeling of subjects (or users) and objects (or rows). If row-level security is considered during the information modeling phase, the appropriate attributes could be implemented as integral parts to the data tables and queries.

In Sybase's Secure SQL Server, access control is enforced through the implementation of sensitivity labels. Sensitivity labels are applied to subject users and procedures, and to object databases, tables, views, and rows. In order to access (i.e., `SELECT`, `INSERT`, `UPDATE`, `DELETE`) any particular object, the subject must possess clearance levels (or labels) that exceed or dominate the object's sensitivity labels.

Labels are composed of two components:

- hierarchical classification
- non-hierarchical compartments

A sensitivity label consists of a single hierarchical classification label plus any number of non-hierarchical compartments. The number of hierarchical classification levels and non-hierarchical compartments is limited only by the secure operating system. Examples of hierarchical classification could be:

- Top Secret
- Secret
- Confidential
- Unclassified

Examples of non-hierarchical compartments could be:

- A - Z

An illustration of label dominance could be:

- Top Secret dominates all Secret, all Confidential, and all Unclassified, but does not dominate Top Secret A.
- Secret A B does not dominate any Top Secret or Secret C, but does dominate Secret A, Secret B, Secret A B, all Confidential, and all Unclassified.

This paper will not attempt to create examples of how sensitivity labels might or should be implemented within a non-secure RDBMS application. It only proposes that the use of such a scheme along with stored procedures and database views could increase the granularity at which row-level and field-level security could be achieved.

3. Database Audit Trails

Auditing is an important part of security in a database management system. Security-related system activity can be recorded in an audit trail, which can be used to detect penetration of the system and misuse of resources. By examining the audit trail, system security officers can inspect patterns of access to objects in databases, and can monitor the activity of specific users. Audit records are traceable to specific users, enabling the audit system to act as a deterrent to users attempting to misuse the system.

This chapter will discuss some database event auditing requirements.

3.1 The Audit System

The audit system should:

- be able to enable and disable system-wide audit options;
- establish auditing of different types of events within a database, or of references to objects within that database from another database;
- establish selective auditing of accesses to tables and views;
- audit the execution of stored procedures;
- audit a user's attempts to access tables and views, or the text of commands that the user executes; and
- allow privileged users to enter user-defined audit records (comments) into the audit trail.

3.2 Establishing Auditing

The System Security Officer should manage the audit system. Only a user who has been granted that role should be able to:

- execute any of the auditing system procedures;
- read the audit trail; and
- access the audit database.

3.2.1 System-Level Audit Requirements

The following is a list of system-level auditing requirements. The system should:

- provide for enabling and disabling of system-wide auditing. A System Security Officer must set the enable auditing option on before any other auditing can take place. Enabling or disabling auditing should automatically generate an audit record, so that you can bracket time periods when auditing was enabled.
- be able to enable or disable auditing of successful, failed, or all login attempts by users.
- be able to enable or disable auditing of all logouts from the server, including unintentional logouts such as dropped connections.

- be able to enable or disable the generation of an audit record when the server is rebooted.
- be able to generate an audit record whenever a user from another host connects to the local server to run a procedure via a remote procedure call (RPC). Auditing should be able to capture all connection attempts, successful attempts only, or failed attempts only.
- be able to audit the use of any privileged commands requiring special roles for execution. It should be able to audit for successful executions only, failed attempts (where failure is due to the user lacking the proper role), or both.
- be able to audit fatal errors (errors that break the user's connection to the server and require the client program to be restarted), nonfatal errors, or both kinds of errors.
- allow privileged users to send text to the audit trail.

3.2.2 Auditing Users

The auditing system should be able to audit a user's attempts to access tables and views in any database, and the text of commands that the user sends to the database server. An access is the use of the `SELECT`, `INSERT`, `UPDATE`, or `DELETE` command on a table or view.

3.2.3 Auditing Databases

The system should permit the establishment of selective auditing on databases. These are some auditable events that can occur in or on a database:

- Use of the `drop`, `grant`, `revoke`, and other table commands within a database.
- Use of the all database level commands.
- Execution of SQL commands from within another database that reference the audited database.

3.2.4 Auditing Tables and Views

The auditing system should allow you to audit accesses of specified tables and views. An access is the use of the `SELECT`, `INSERT`, `UPDATE`, or `DELETE` command on a table or view. It should allow the establishment of auditing of specified tables and views, or create default audit settings for newly-created tables and views.

3.2.5 Auditing Stored Procedures

The auditing system should allow you to audit the execution of stored procedures. The values of any parameters passed to a stored procedure should be audited. It should allow the establishment of auditing of existing stored procedures or create default audit settings for newly-created stored procedures.

3.2.6 Adding User-Specified Records to the Audit Trail

The system should allow privileged users to enter user-defined audit records (comments) into the audit trail. For instance, it may be important to note with comments such actions as disabling and enabling particular types auditing.

3.3 Audit Trail Operations

The only operations allowed on the audit trail should be SELECT and other commands that may be executed only by a system security officer.

3.4 Archiving Audit Data

Because the audit trail is appended continuously, it is necessary to archive old audit data from time to time, depending on the size of the device on which it resides. If the audit device fills up, audited system activity and database activity may come to a halt.

4. Summary

The user authentication process provided by the DBMS vendors must be considered vulnerable to attack. It may be necessary to investigate the effectiveness of using a DCE/Kerberos system accompanied with smart card technology to provide a safe authentication process (see the appendix on DCE and the chapter on Smart Cards).

This chapter has presented several examples of stored procedures and database views to provide and enforce row-level security. It has proposed that the addition of a sensitivity labeling scheme could be added to increase row-level and field-level security granularity. Oceania's scheme of granting permissions to tables to users and groups of users alone will not provide row-level security. Legitimate users and system administrators will be able to access all rows in tables where permissions have been granted. They will be able to access the database tables through third party query tools bypassing the client application.

Stored procedures offer many advantages, but there are some drawbacks. The main drawback of stored procedures is that they are non-standard. No two vendor implementations are alike. The language for describing the stored procedures and their functionality vary from server to server. Stored procedures are not portable across vendor platforms. There is no standard way to pass or describe the parameters.

In a typical two-tiered application such as Oceania's application, the business rules are enforced in either the client application, the stored database procedures, or some combination of the two. In a three-tiered architecture (see the appendix on Multi-tiered Architecture) that contains a data layer, a functional layer, and a presentation layer, the business rules are enforced in the functional layer residing on a server. The stored procedures presented in this paper along with the rules of the business could be implemented in a functional layer. Implementing a functional layer would avoid the drawbacks of stored procedures and remove functions from the client application. Authorizations to execute the functions would have to be granted and maintained through a facility independent of the client application and the database management system such as the DCE's Privilege Service and Access Control Lists (ACLs) (see the appendix on Distributed Computing Environment). The Privilege Service determines whether a given client is authorized to perform specific operations. Access to the database would be granted to the functional layer only. Direct access would not be granted to users or groups.

In order to meet the requirement that a "user's rights are identical whether the interaction is through the Electronic Medical Record (EMR) application or third party SQL tools"[1], it will be necessary to implement a sophisticated security mechanism that will provide and enforce user privileges regardless of access method. Granting permissions to access tables and views to users and groups of users alone will not suffice. It is suggested that direct access not be granted to tables and that some combination of views, sensitivity labeling, and stored procedures or functions be used to accomplish row-level and/ or field-level security. There will be maintenance issues involved with a sophisticated security scheme. Privileges to views and stored procedures would have to be managed through the DBMS. Tools would probably have to be developed to managed the sensitivity labeling of subjects and objects.

Auditing is an important part of security in a database management system. An extensive audit system is required to detect penetration of the system and misuse of resources. By examining the audit trail, system security officers can inspect patterns of access to objects in databases, and can monitor the activity of specific users. Audit records are traceable to specific users, enabling the audit system to act as a deterrent to users attempting to misuse the system. Most DBMS vendors supply auditing capabilities that may be sufficient for recording database activity.

1. Oceania Inc., *Security Functional Requirements V.2.2*, January 30, 1996

Chapter 5 - Smart Cards

A smart card contains a microcontroller that is mounted onto a plastic card the size of a credit card. The card has specific built-in functions such as protected memory, cryptographic functions, and I/O control. The microcontroller contains a microprocessor usually based on the Motorola 68HC05 or Intel 8051 calculation unit (used for the encryption algorithms), small amount of RAM (usually around 256 bytes), ROM which is used to store the controller's operating system and instruction set (from about 4K to 14K bytes), and user memory in the form of EEPROM. Currently the EEPROM ranges from 256 Bytes to 8K bytes depending upon the application needs and budget allocated to pay for the cards (current prices are from $4.00 to $19.00 for small quantities of cards). The encryption algorithms are usually DES, RSA, or both.

A card's memory basically determines the card's performance and usability. The size of the RAM greatly affects the encryption performance. The size of the ROM determines what functions can be included in the card's operating system. The EEPROM determines how much user or application information can be stored in the card.

Smart card operating systems can be rather difficult to understand. In addition, the technical documentation for the card's operating system is often poorly written. Smart card application developers will need to become very familiar with the operating system for the respective cards they are working with.

Some of the advantages of smart cards are that they:

- Can easily be used as an ID badge;
- Perform basic cryptographic functions that can be used to provide authentication and digital signatures;
- Provide protected memory that can be used to store a user's private encryption keys and other confidential information; and
- Are inexpensive technology.

A major disadvantage of smart cards is the small amount of usable memory (only a few thousand bytes).

1. Standards

The International Standards Organization (ISO) has defined several standards for smart cards, which include:

- ·ISO 7816-1. Physical characteristics.
- ·ISO 7816-2. Dimensions and location of contacts.
- ·ISO 7816-3. Electronic signals and transmission protocols.
- ·ISO 7816-4. Inter-industry commands and responses (i.e., read, update, etc.).
- ·ISO 7816-5. Registration system for applications in cards (standard method to store application `XYZ' on a card).
- ·ISO 7816-6. Data Elements for interchange (i.e., user's name, address, card expiration date, etc.).

2. Current Uses

While smart cards are being utilized heavily in Europe and Asia, they are currently still in the early stages of development in the U.S. Some examples of their current uses are given below.

- Pay Telephone Debit Cards. Individuals purchase a fixed amount of telephone credits. When they use the card, the phone deducts the number of credits consumed by the call from what is available on the card. The card is disposed of when all the credits are used.
- Electronic Purse. Individuals can go to their local bank or ATM and transfer funds from their account into their smart card. The card is then used like cash. The Marine Corps training facility at Paris Island is currently using a modified version of the electronic purse. Recruits have their pay entered in the cards which they then use to make purchases on the base or get cash withdrawals from post exchanges.
- Vehicle Maintenance. Toyota in Japan is issuing a smart card with each new car to track the vehicle's maintenance history.
- Agriculture. The U.S. Department of Agriculture has issued smart cards to several peanut farmers to monitor and control crop sales.

3. Common Features

3.1 File Structure

The basic file structure on the card is divided into three areas, as shown in the structure below. The first is the Master File (required); the next level is the Dedicated File (DF) and finally, the Elementary File (EF). DFs are used to store EFs (like a sub-directory in DOS) and the EFs store user data (like user files in DOS).

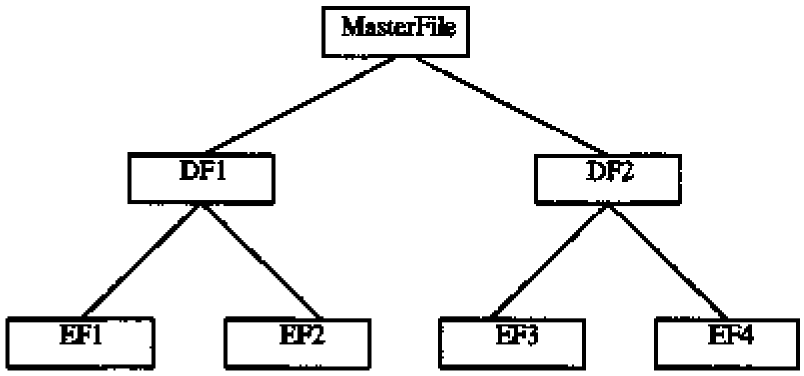


Figure 3.1 Smart Card File Structure

3.2 Security

Security is usually handled by the use of a Personal Identification Number (PIN). The PIN, however, is not limited to numbers, but can be any ASCII character (so the PIN can be a password). A maximum of eight characters can be used to create the PIN.

Once the PIN has been correctly submitted, the card then will allow access to the data protected with that PIN. For example, EF1 and DF1 files could be protected by a single PIN. Once that PIN is entered, all data in those areas can be accessed by the user. DF2 on the other hand could be protected by a different PIN. Many cards have variations on this approach where EFs within a specific DF can be individually protected by their own PIN. This can be useful if several different applications wish to use the same card. If the card is to be used in only one application, this feature is often not used.

In order to protect the data in the card from password attacks, the card will only allow a fixed number of wrong PIN submissions by the user. Usually this value can range from one to seven and can be programmed by the application developer. If a user submits incorrect PINs beyond the specified limit, the card will become locked until the card issuer unlocks it. For example, if the value is three then after three wrong submissions the card is locked until the issuer unlocks it. With some cards the three wrong entries must be entered consecutively (three misses in a row). With other cards, wrong PIN entries are cumulative (three wrong submissions over the life of the card). To unlock a card, the issuer must usually submit its PIN and the user's PIN (this can be a problem if the user has forgotten his or her PIN).

3.3 Communication

Communication with a smart card is performed via a serial I/O port. The I/O speed of most smart cards is 9600 bits per second.

3.4 Instruction Set

Smart cards have a limited basic instruction set due to the fact the operating system must fit into a few thousand bytes of ROM. Each smart card vendor has a slightly different instruction set but most include the following functions:

- Read data stored in the card's memory.
- Write data to the card's memory.
- Erase data stored in the card's memory.
- Submit PIN or key.
- Read result of last operation performed by the card (for example, result of DES encryption).
- Generate a random number.
- Select DF or EF.
- Create DF or EF.
- Generate temporary DES key (i.e., session key).
- Encrypt (use DES to encrypt an eight byte block of data).
- Decrypt (use DES to decrypt an eight byte block of cipher text).
- Increment/decrement count.
- Increase/decrease the balance of an electronic purse.

Most card manufacturers provide a set of high level language libraries in C. Some vendors also provide Visual Basic libraries.

4. Cryptographic Uses of Smart Cards

Microprocessors with built-in cryptographic capabilities such as DES, RSA, and DSS (Digital Signature Standard) are available. Smart cards with cryptographic capabilities accessible by application developers can be used in key management, strong user authentication, encryption, and digital signatures.

In a typical Kerberos or DCE environment, a user submits his or her password (shared by the user and the Kerberos server) to a client. The client uses the user's password to encrypt/decrypt authentication information to/from the Kerberos/DCE server. The use of a smart card in a Kerberos/DCE environment has four advantages, as described below.

- The user's Kerberos password can be securely stored on the card and does not have to be remembered by the user.
- Since the user does not have to remember his or her password, the password could be the 56-bit DES key shared by the Kerberos/DCE Server.
- Since the password is stored on the card, it could be used to decrypt and store the Kerberos/DCE credentials.
- The password is never passed to the client.
- It provides two-factor authentication. The user must use a password to gain access to the smart card and must have physical possession of the card to gain access to the Kerberos password.

Smart cards with support for public key cryptography can also be used for digital signature generation. The card can store the signer's private key and compute the digital signature. As such, the signer's private key is not exposed to the client workstation. In order to improve the performance of document signing, the client workstation can perform the one-way hash function on the data that is to be signed. The hashed data can then be passed to the card, which will then compute the digital signature.

5. Functional Requirements

An examination of ten smart card vendors was performed for this CRADA. The types of smart cards available from these vendors range from protected memory cards which have no processing capabilities to cards that have complex cryptographic capabilities. To be able to use the smart cards as described above in a Kerberos or DCE environment, the smart card must meet the following functional requirements.

- The System Security Officer must be able to choose and store the user's Kerberos/DCE password (a DES key) in the card's protected memory. This is necessary so that both the card and Kerberos/DCE server share the same key, and that it can be updated when necessary.
- The card must be support public key cryptography for digital signatures.
- The card must be capable of using the DES key(s) stored in its protected memory to perform encryption and decryption of data. The card must be able to decrypt the Kerberos/DCE ticket granting ticket on the card.
- The card must have a basic file system that will be used to store information deemed necessary by application developers.

- The card vendor should provide a high level language library (for example, C) for the card. If this is not provided the application developers will have to program the card with machine language.
- The card vendor must provide good technical documentation and good technical support.

Microprocessors with cryptographic features are available that would enable smart cards to be used for key management, strong user authentication, encryption, and digital signatures. However, no COTS smart card with such features has been identified. Custom smart cards could be manufactured, but at an extra expense. Hence, one must consider the individual strengths and weaknesses of each card and compare them to the minimum requirements for an application. In the near term, vendors are expected to develop smart cards capable of meeting all of the specified requirements. Such cards may become available in 1997.

Chapter 6 - Architecture Alternatives

1. Introduction

After assessing the surety needs of Oceania, this project focused on Commercial Off-The-Shelf-based (COTS) solutions. The hope was that COTS-based solutions would minimize the amount of surety development by Oceania. However, the current state of COTS surety products and technologies replaces custom surety development with custom and complex integration and surety system administration. No product today provides the complete set of surety services required by systems like Oceania's. The best that can be done today is to integrate several products to provide the services needed. There are trade-offs associated with each combination of products. The architectures below describe specific alternatives built using different combinations of products and the issues that must be considered with each.

2. Specific Architecture Alternatives

Each of the specific architectures described in the following sections is based on the generic architecture shown in Figure 2.1. Custom and COTS applications are supported by a *surety* layer which in turn runs on a COTS operating system. The differences between the various architectures are limited to the surety layer. The architectures described are peer-to-peer indicating that various peers may act as servers for different applications. The specific architectures include only technologies that are currently available or which are known to be available in the near term.

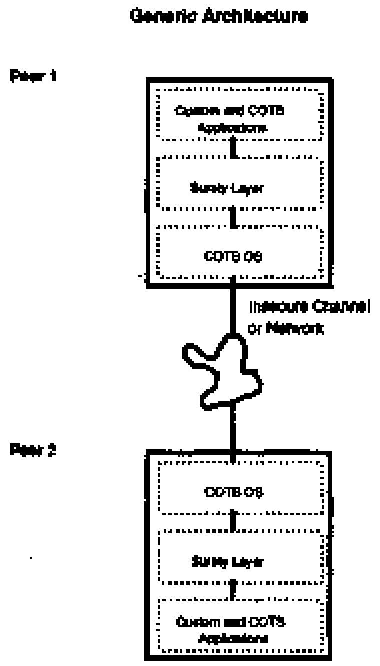
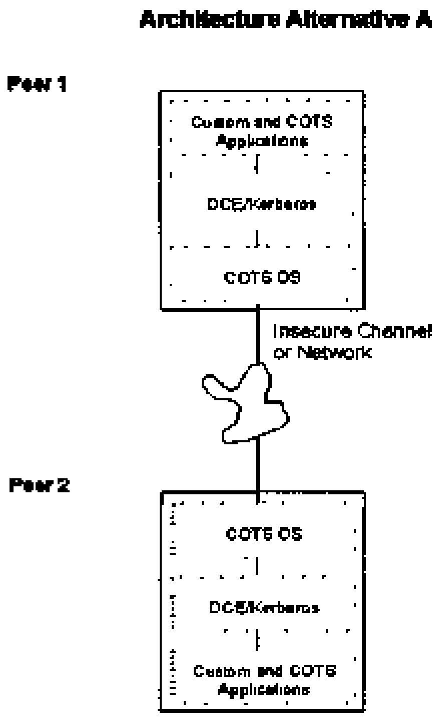

Generic Architecture
Peer 1
Custom and COTS Applications
Surety Layer
COTS OS
Insecure Channel or Network
Peer 2
COTS OS
Surety Layer
Custom and COTS Applications

Figure 2.1 Generic Architecture

The following sections describe each alternative and discuss the surety risks mitigated and those not addressed by the architecture. In addition, the technologies described in the chapter on Basics of Information Surety are considered where applicable.

2.1 Architecture Alternative A

Architecture Alternative A is shown in Figure 2.2. The surety layer functionality is provided by DCE [OSF, 1992, 1996] (see appendix on DCE) or Kerberos [Neuman, 1993, 1994] [Jaspan, 1995]. DCE and Kerberos are both surety middleware products.


Architecture Alternative A
Peer 1
Custom and COTS Applications
DCE/Kerberos
COTS OS
Insecure Channel or Network
Peer 2
COTS OS
DCE/Kerberos
Custom and COTS Applications

- **Services**
  - User authentication
  - Access control
  - Communications privacy
- **Issues**
  - Heavy maintenance and administrative burden associated with DCE
  - DCE provides a distributed computing environment which includes more than security features
  - DCE provides a distributed time service and audit trail capabilities
  - Some customers may be adopting a DCE infrastructure
  - No public key support
    - No digital signatures
  - Application code must be altered
    - Impact of code modification is greater when using DCE
  - Peer which acts as security server must be physically protected
  - Peer which acts as database server must be physically protected
  - Smart cards may be used

Figure 2.2 Architecture Alternative A

DCE security is based on Kerberos so many of the surety features in the current version of DCE are provided by the underlying Kerberos libraries. In particular, Kerberos furnishes the authentication capability. However, DCE supplies other functionality in support of distributed computing including audit trail and a distributed time service capability. While DCE supports flexible access control to system objects, access control within an application must still be managed by the application. DCE does provide Access Control List (ACL) (see the appendix on DCE) support but the lists must be managed by the applications via ACL management software written by developers. Confidentiality across communications links is a feature of both DCE and Kerberos. However, encryption is used to provide this feature and so export controls apply to both of these products.

DCE is a complex environment so the system administrative and maintenance costs could be quite high. Since Kerberos provides some of the underlying surety features for

DCE, it appears likely that a system run on M.I.T. Kerberos version 5.0 will be compatible with a system run on DCE version 1.1. Kerberos requires a much smaller system administration burden and might therefore, be the better choice for Oceania to support.

Neither Kerberos nor DCE provide any digital signature capabilities. Open Software Foundation (OSF), the consortium that sponsors the development of DCE, has formed a working group to develop a public-key infrastructure. However, it does not seem likely that a public key infrastructure capability will appear in the near term. Since digital signature algorithms are public-key algorithms, a public-key infrastructure that provides key management functions is required. While the availability of digital signature capabilities in DCE may not be imminent, OSF is aware of the need. DCE does support the use of smart cards within the system and smart cards can be used to generate digital signatures [Merckling, 1994]. However, there are several caveats:

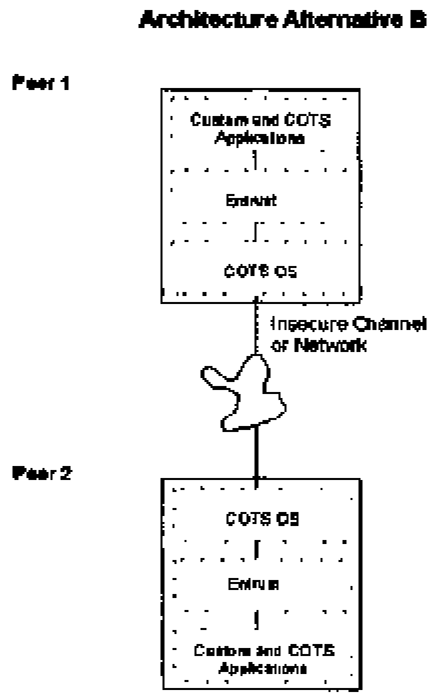
- applications must support smart cards;
- principal registry structures must be changed;
- some protocols must be changed (login, account creation, password and key modification); and
- the smart card must be able to store auxiliary files.

However, digital signature capabilities generally require a public-key infrastructure that would have to be developed.

DCE and Kerberos require modifications to application code to make use of their surety features. Therefore, portability to systems other than those which are DCE- or Kerberos-enabled must be addressed. Within the system, peers that act as database servers and/or security servers must be physically protected. This protection could be provided by simply locking these servers in a room and limiting access.

2.2 Architecture Alternative B

Architecture Alternative B is shown in Figure 2.3. The surety layer functionality is provided by Entrust from Northern Telecom [Entrust]. Entrust features may be accessed through the Entrust Programmer's Toolkit or via the Entrust application.


Architecture Alternative B
Peer 1
Custom and COTS Applications
Entrust
COTS OS
Insecure Channel or Network
Peer 2
COTS OS
Entrust
Custom and COTS Applications

- **Services**
  - User authentication
  - Digital signatures
  - Encryption
  - Key management
  - Public key infrastructure
- **Issues**
  - No access control
  - Peer which acts as security server must be physically protected
  - Peer which acts as database server must be physically protected
  - User is associated with a particular peer
  - No smart card support
  - Application code must be altered
  - Some customers may be adopting a DCE infrastructure

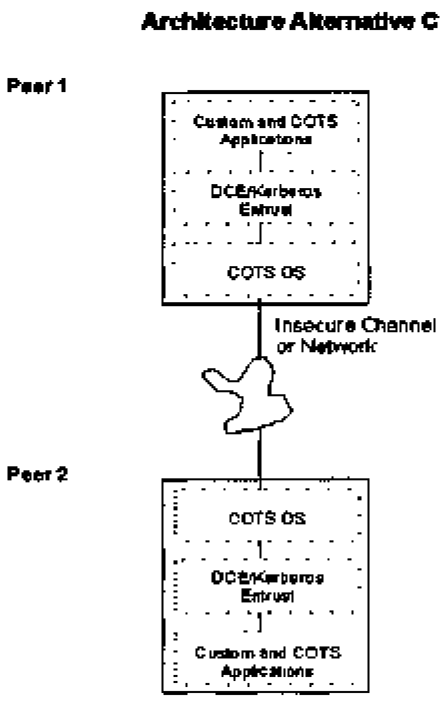
Figure 2.3 Architecture Alternative B

This architecture provides a public-key infrastructure that includes X.509 certificates [ITU-T, 1993]. These certificates are internationally recognized which would allow for broad compatibility with other such applications. Entrust supplies both RSA and DSA digital signature algorithms. Encryption is available to provide confidentiality of data in storage and transmission. Export controls clearly apply. Entrust manages both encryption and digital signature keys. However, Entrust provides no support for smart cards. This lack of support would not be problematic except that Entrusts key management mechanism associates a user with a particular peer. Handling users who access the system from various peers would be difficult although this issue might be mitigated by development to support smart cards independent of Entrust.

Entrust supplies user authentication functionality but not access control functionality. In addition, using Entrust alone would not support a DCE-based architecture. As with DCE and Kerberos, application code would have to be altered to make use of Entrust's surety features so portability issues would have to be addressed. However, Entrust is IDUP-GSS-API [Adams, 1996] compliant so applications can be ported to other IDUP-GSS-API compliant environments. In addition, the peers that act as security servers and database servers must be physically secured.

2.3 Architecture Alternative C

Architecture Alternative C is shown in Figure 2.4. The surety layer functionality is provided by DCE or Kerberos, and Entrust. This architecture combines the access control capabilities of DCE/Kerberos with the digital signature, key management and public-key infrastructure capabilities of Entrust. In addition, this architecture would support a DCE-based architecture. Not unexpectedly, integration of these products would require some effort.



Architecture Alternative C
Peer 1
Custom and COTS Applications
DCE/Kerberos Entrust
COTS OS
Insecure Channel or Network
Peer 2
COTS OS
DCE/Kerberos Entrust
Custom and COTS Applications

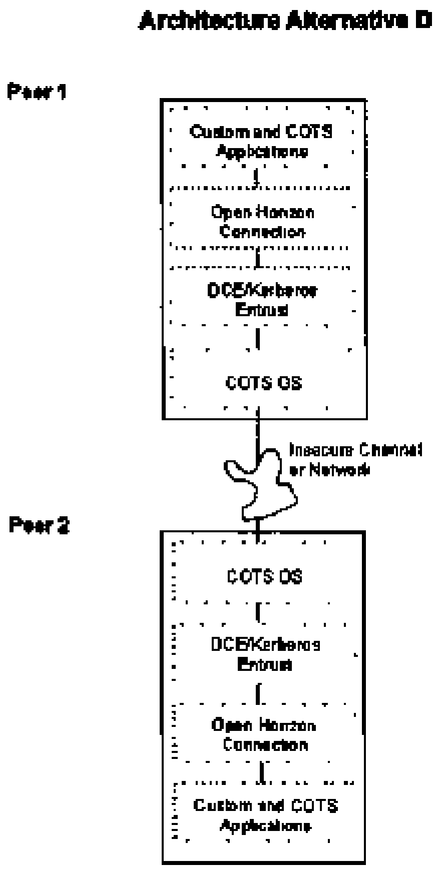
- **Services**
  - User authentication
  - Access control
  - Digital signatures
  - Encryption
  - Key management
  - Public key infrastructure
- **Issues**
  - Heavy maintenance and administrative burden associated with DCE
  - DCE provides a distributed computing environment which includes more than security features
  - DCE provides a distributed time service and audit trail capabilities
  - Some customers may be adopting a DCE infrastructure
  - Application code must be altered
    - Impact of code modification is greater when using DCE
  - Peer which acts as security server must be physically protected
  - Peer which acts as database server must be physically protected
  - Smart cards may be used
  - User is associated with a particular peer for the purposes of generating digital signatures

Figure 2.4 Architecture Alternative C

If DCE is chosen rather than Kerberos, the system administration and maintenance burden is quite high. Application code would have to be modified to make use of DCE/Kerberos and Entrust surety features; so code portability is an issue. Smart cards could be used for user authentication and encryption, but users would be associated with a particular peer for purposes of generating digital signatures. Although, this issue might be mitigated by development to support smart cards independent of Entrust. See a discussion of the caveats for using smart cards with DCE in the section on Architecture Alternative A. Finally, peers that act as security servers and database servers must still be physically secured.

2.4 Architecture Alternative D

Architecture Alternative D is shown in Figure 2.5. The surety layer functionality is provided by DCE or Kerberos, Entrust and Open Horizon's Connection [OpenHorizon] (see the appendix on Open Horizon's Connection). In addition to the functionality provided in Architecture Alternative C, Connection provides a single sign-on capability and application authentication. Single sign-on in this context means that the user is not forced to log on to the system and then log on to various applications separately. The user logs on only once. Application authentication means that applications can authenticate themselves to the system and to other applications. Perhaps more importantly, the use of Connection would allow developers to easily write portable applications. Connection currently supplies support for DCE and Kerberos. Support for Entrust is under development.



- **Services**
  - User authentication
  - Access control
  - Digital signatures
  - Encryption
  - Key management
  - Public key infrastructure
  - Single sign-on
  - Application authentication
- **Issues**
  - Heavy maintenance and administrative burden associated with DCE
  - DCE provides a distributed computing environment which includes more than security features
  - DCE provides a distributed time service and audit trail capabilities
  - Some customers may be adopting a DCE infrastructure
  - Peer which acts as security server must be physically protected
  - Peer which acts as database server must be physically protected
  - Smart cards may be used
  - User is associated with a particular peer for the purposes of generating digital signatures

Figure 2.5 Architecture Alternative D

The maintenance and administration of DCE is still burdensome and peers that act as security servers and database servers must still be protected. Users are still associated with a particular peer for the purposes of generating digital signatures. Although, this issue might be mitigated by development to support smart cards independent of Entrust. See a discussion of the caveats for using smart cards with DCE in the section on Architecture Alternative A.

Appendix A - Multi-tiered Architecture

1. Introduction

Multi-tiered application architecture, sometimes referred to as application partitioning, is not a new concept. Application designers have attempted to partition applications in one form or another for many years. Business logic has been separated from data access in many corporate applications in order to allow for future changes in technology. Applications developed in client/server environments are divided into components and typically distributed to two or more computers. Client/server physically distributes process, data, and transactions across local and wide area networks.

2. Two-tiered Model

A two-tiered application usually consists of two primary machines: the user's client machine and a remote database or file server. In a typical two-tiered application, the business rules are enforced in either the client application (or user interface), the stored database procedures, or some combination of the two.

When two-tiered departmental applications are pushed to support enterprise use, organizations encounter a number of problems. These problems include unacceptable performance, lack of scalability, high costs of maintenance, inability to share business rules across applications, and the inability of the first-generation client/server tools to handle highly complex business logic. Additionally, organizations find that they lack some of the mainframe quality services such as security services, which include user authentication, authorization and data protection, centralized directory services, and a coherent management solution.

Organizations have invested heavily in the development of logic that encapsulates rules about the business. With these business rules located on client workstations embedded within an application, they are inaccessible to other applications. Therefore, they must be re-developed and maintained for each individual application.

3. Three-tiered Model

Client/server application partitioning scenarios have been worked out in detail by several organizations. The most popular partitioning scheme divides an application into three specific layers: the presentation layer, the function layer, and the data management layer. Developers accommodate the presentation layer by creating a user interface. The function layer is refined into subcomponents including data validation, data integrity, and transaction-processing control, and apportioned between the client and server. The data management layer accommodates the data services, such as the database management system, on one or more servers.

There are some inconveniences to partitioning applications. One is complexity. The distribution of process, data, and transactions across numerous physical environments greatly increases complexity. The application's components no longer reside within a single, well-controlled physical environment. They must now communicate among themselves in a coordinated fashion. A complex transaction may have to access several computers in order to complete as a logical unit of work.

Another inconvenience is connectivity. A client/server application adds to the competition among applications by distributing them across networks with finite capacities. Developers must overcome this issue by optimizing distribution for performance.

Distributed applications bring considerable flexibility and challenge to the developer's world. It is considered normal for part of an application to exist on the end-user's desktop while attached to a local area network. It is also expected that the same functionality be portable. The application's end user may need to access the application from home or on the road using a laptop computer and a means of connectivity that's not nearly as fast as a departmental local area network. A specific connectivity scenario may require partitioning the application in a way that further adds to its complexity. There is also a need to insulate the required application partitions from the end users, making them transparent.

4. Choice of Tools

The choice of tools to implement a three-tiered architecture covers a broad spectrum. Beginning with the presentation layer, several popular 4GL integrated development environments are on the market. 3GL languages such as C/C++ and COBOL may also be used.

With the business (or function) layer, the developer has even more choices. 4GL development is an option for the function layer; however, when combined with the 4GL of the presentation layer, it leads to the fat-client implementations that are prevalent today. In addition, traditional 3GL languages such as C/C++, COBOL, or FORTRAN may be used. Notice also that RDBMS proprietary stored procedures are also used to implement the functional layer. However, stored procedures are non-standard but in this architecture, the functional layer could be implemented to perform the tasks of stored procedures using ANSI SQL. This would provide portability between commercial database management systems.

The biggest challenge is deciding the physical location of service execution. A direct relationship exists between the physical tools chosen to build logical services and where the components can physically execute. For example, if functional layer services are constructed using a 4GL tool such as VisualBasic, then the developer's only choice for execution is on the processor serving the client. A better solution would be to write as much of the business logic as possible using a tool that provides the most flexibility, portability, and leverage for the future, like C/C++ and COBOL.

Appendix B - Distributed Computing Environment

In 1990, the Open Software Foundation (OSF) announced the Distributed Computing Environment (DCE), a comprehensive, integrated set of services that supports the development, use and maintenance of distributed systems. DCE is a distributed architecture that consists of Security Services, Directory Services, Remote Procedure Calls (RPCs), Threads, Distributed File System, and Time Services. DCE is an industry standard and its specifications have been incorporated by X/Open into the Common Application Environment. DCE allows companies to create an environment in which all systems and their resources are immediately accessible to users on the network, regardless of their location, and provides the necessary services across multiple operating environments for security and global naming.

1. Introduction to DCE

Computer users require a communications environment that will allow information to flow from wherever it is stored to wherever it is needed, without exposing the network's complexity to the end user, system administrator or application developer. The goal of distributed computing is to make a network of individual computers act as one. Benefits include uniform access to resources and ease of resource sharing. The OSF DCE is a software system that allows people to build scalable, secure distributed applications.

DCE provides an architecture for building applications that are:

- Scalable,
- Secure,
- Distributed,
- Interoperable with other resources, and
- Portable across heterogeneous platforms.

DCE comprises the following components:

- Secure Core:
    - Cell Directory Service (CDS),
    - Global Directory Agent (GDA),
    - Remote Procedure Call Facility (RPC),
    - Security Service,
    - Distributed Time Service (DTS), and
    - Threads Facility;
- Extended Services:
    - Global Directory Service (GDS), and
    - Distributed File Service (DFS); and
- Others:
    - Audit Services, and
    - Server Management Facilities.

1.1 UUIDs

DCE makes abundant use of numeric identifiers called Universal Unique Identifiers (UUIDs) for anything that requires a unique identifier (i.e., DCE cells, users, processes, interfaces, etc.). A UUID is built from a timestamp, process ID of the generating process, a machine ID, and 32 bits for uniqueness. UUIDs have no inherent meaning.

An example UUID:

```
00162b64-6738-1060-933e-9e621066aa77
|        |         |    |
|        |         |    Machine ID
|        |         Process ID
|        Timestamp
Uniqueness bits
```

1.2 DCE Cells

DCE divides its world into cells. A cell is a collection of users, computers, and other resources managed as a group. Boundaries depend on the purpose of the cell. A cell may encompass an entire organization, departments within an organization, or some other logical division.

Each DCE user and machine belongs to a single cell, called the local or home cell. Other cells are called foreign cells.

Each cell includes components of the DCE Secure Core:

- Remote Procedure Call Facility (RPC),
- Cell Directory Service (CDS),
- Security Service,
- Distributed Time Service (DTS), and
- Threads Facility.

1.3 Remote Procedure Calls

The OSP Remote Procedure Call capability is based on the premise: make individual procedures in an application run on a computer somewhere else in the network. RPC is DCE's mechanism for inter-process communication. An RPC is a function call that looks like an ordinary local function call. The calling code and called procedure actually belong to different processes that may be on different machines. The client program makes the RPC and the server program answers the RPC. A process may be both a client of some server and a server to some clients. It should be noted that DCE RPCs are not necessarily compatible with other vendors' RPC facilities.

1.4 DCE Servers and Application Servers

The phrase "DCE server" correctly describes both a DCE server and/or an application server.

DCE's system servers implement DCE services (i.e., Security servers, Name servers, etc.). They are provided by the vendor and may act as servers to other system servers and application servers.

Application servers implement particular applications, are written by a site's programmers, and are often clients of DCE's system servers.

1.5 RPC Interfaces

An interface is a set of function descriptions. A server may offer one or more interfaces. Many servers may offer the same interface. Each interface is defined in a separate file, is written by a programmer, some information is compiled into clients, and some information is compiled into servers. The interface definition file includes the UUID and version number, the interface name, and the set of function prototypes.

1.6 The DCE Control Program (dcecp)

DCE 1.1 provides a unified administration tool, `dcecp`, that supports most, but not all, of the DCE administration functions. It is based on Tcl (Task Control Language), version 7.3. `dcecp` attempts to provide a common, consistent command-line interface to all DCE components for user interaction and writing scripts and programs.

2. Directory Service

The Directory Service provides a single naming model throughout the distributed environment. It allows the users to identify by name resources such as servers, files, disks, or print queues, and gain access to them without needing to know where they are located in a network. Sometimes called the Naming Service, the Directory Service offers a general way to locate distributed resources.

DCE client applications will usually know:

- the server's CDS name: indicates the DCE cell and server desired;
- the RPC interface specification: indicates the functions offered by a server; and
- possibly, an RPC object UUID, associated by the server code with some specific resource.

Programmers determine how clients acquire this information.

To communicate with servers, DCE clients need to find three pieces of data:

- the machine on which the server runs (i.e., IP address);
- the protocol sequence to use when communicating with the server (e.g., TCP/IP or UDP/IP); and
- the protocol-specific endpoint (i.e., a TCP/IP port number).

Clients store this information in data structures called binding handles.

Directory services fall into three categories:

- Global name services which:
    - indicate the home cell of a server, and
    - allow clients to narrow their searches to specific DCE cells;
- Cell-wide services which:

        - indicate the host machine of a server, and
        - allow clients to narrow searches to specific machines;
- Machine-specific services which:
    - indicate the precise location of a server, and
    - allow clients to pinpoint a server.

2.1 Cell Directory Service

The Cell Directory Service (CDS) is DCE's cell-wide directory service. CDS maps server's DCE names to locations. It does not know about foreign cells, does not map host names to addresses, and is not a replacement for `/etc/hosts` or DNS. It is, also, known as the name service.

CDS servers maintain the CDS database. CDS databases are called clearinghouses.

Each DCE cell has a namespace which represents its resources that is structured as a tree. Servers have names like: "/.../Oceania.com/medDB". The full name indicates the DCE cell. Client programs pass this name to CDS in return for the server's location.

2.2 Global Directory Agent

CDS understands names in the local cell. Other software is used for intercell communications. DCE provides the Global Directory Agent (GDA) as an intermediary between a cell's CDS and the remote name services.

If a client wants to contact a remote server:

- The client's GDA uses DNS to look up the other cell,
- DNS tells GDA where the other cell's CDS server is,
- GDA tells the client where to find the other cell's server, and
- The client asks the other cell's CDS for the servers location.

2.3 Endpoints

After finding the server's machine through CDS, the client must find the endpoint(s) at which the server is listening. "Endpoint" is a generic term for a specific network address (for example: a port in TCP/IP).

The RPC endpoint mapper:

- Tracks which servers are currently using which endpoints (ports),
- Runs on every machine,
- Maintains a local table that is:
    - the endpoint database (or map), and
    - a per-machine, not per-cell, table:
        - is more dynamic than IP addresses, and
        - is of no use elsewhere.

2.4 Putting It All Together

When starting, a DCE server:

- DCE informs the endpoint mapper of the ports it's using,
- informs CDS of:
    - the IP address of the host,
    - any supported protocol sequences, and
    - services provided:
        - interfaces, and
        - RPC objects.

A DCE client contacting a server imports binding information by:

- finding a foreign cell with global naming and GDA (if necessary),
- locating the machine through CDS,
    - the machine IP address,
    - the protocol sequence;
- determining the server's endpoint, and
- communicating with the server.

2.5 Access Control

CDS protects its resources with Access Control Lists (ACLs). ACLs provide privileges to user requests based on DCE Security Identities (principals). The CDS server stores and manipulates ACLs. Administrators manage ACLs via the DCE Control Program ACL manager.

CDS places ACLs on everything:

- Names in the namespace are for:
    - Directories,
    - Objects,
    - Clearinghouses,
    - etc.
- Processes
    - DCE client process
        - CDS advertiser - a per-host process that creates and loads the host's CDS cache, locates CDS servers at DCE start-up, and creates cdsclerk processes for users.
        - CDS clerks - a process for each CDS user on a machine that makes requests to the CDS server for importing client binding information and exporting server binding information.
    - CDS server process: cdsd is a process that runs on a DCE server that maintains the CDS database, writes from servers exporting information, reads from clients importing information, etc.

2.6 CDS Replication

Replication makes CDS information more available by making copies of the information.

- A Replica is a physical copy of a CDS directory where a writable copy is called the master and a read-only copy is called read-only.
- A Clearinghouse is a collection of directories where a cell can have multiple clearinghouses, each holding some subset of directories in the namespace.
- A replication unit is a CDS directory.

Each CDS directory can be stored in multiple clearinghouses. In the most common form of replication, one clearinghouse holds the master replica and other clearinghouses hold read-only replicas. CDS clients automatically talk to the proper kind of replica: master for updates read-only or master for lookups.

Skulking is when CDS automatically propagates updates from the master to the read-only replicas.

2.6.1 Multiple Clearinghouses

Replication results in multiple clearinghouses. Each is maintained by a `cdsd` process. Each has files stored on local disks. Each has an entry in CDS.

Each clearinghouse may hold a different subset of CDS directories. Depending on the replication scheme, a clearinghouse may hold master copies of some directories and read-only copies of others.

2.6.2 Multiple `cdsd` Processes

Multiple clearinghouses mean multiple CDS servers, each of which exports binding information to CDS.

3. Security Service

Security servers offer three component services:

- Registry Service - manages information about DCE users,
- Authentication Service - verifies the identities of users, and
- Privilege Service - helps determine which users are allowed to do what.

Some Security-related definitions:

- Principal: any user of DCE services (i.e., people, computers, processes, cells).
- Account: information used when a principal logs in is:
  - similar to UNIX /etc/passwd file, and
  - includes a password, home directory, etc.
- Group: a collection of principals used for access control, and
- Organization: a collection of principals assigned a set of password rules.

3.1 Registry Service

Each cell has its own Security Registry that holds information about DCE principals and accounts, security groups, and other information. It is separate from the operating system's security files (like UNIX's `/etc/passwd` and `/etc/group`).

3.1.1 Security Assumptions

DCE Security works under these assumptions:

- Machines on users' desks are untrustworthy.
    - Users are knowledgeable and malicious.
    - Hardware and software can be hacked.
- Cell Directory Service is untrustworthy.
- Only Security server machines are trustworthy.
    - Hardware and software are safe.
    - Bad guys can't get administrative access.
- All DCE users keep their passwords secret.

3.1.2 Principals

A principal identifies a user of DCE services as any of the following:

- people,
- machines,
- processes, and
- DCE cells.

DCE services include:

- Cell Directory Service (CDS),
- Distributed Time Service (DTS),
- Distributed File System (DFS),
- Security Service, and
- access to other applications using such services.

3.1.3 Accounts

The Security Registry also holds an account for every DCE user. A principal without an account cannot log into DCE.

Each account is a "triple" of:

- Principal,
- Principal's primary DCE Security group, and
- Principal's primary DCE Security organization.

Before an account can be created, each element of the triple must exist and the principal must belong to both, a group and an organization.

3.1.4 Security Groups

Security groups are used only to simplify access control. A set of principals with similar rights belongs to the same primary group. Principals may belong to many groups.

3.1.5 Organizations

An organization holds a collection of principals similar in administration to a Security group. It is used for password management, not access control as groups are. Each principal belongs to a "primary" organization.

3.1.6 Policies

Policies are sets of rules governing security. Different policies may apply to the entire cell, each organization, and individual accounts.

3.1.7 Aliases

A user can have multiple, alternate names, "aliases". One principal is called the primary. One primary may have multiple aliases. Aliases allow a single user to assume multiple DCE identities that share the same UUID. Each alias has a separate account with a different password, different group, and different organization. DCE's Distributed File System (DFS) allows aliases to access different sets of files.

3.2 Authentication Service

The Authentication Service allows users to establish DCE identities. Authentication proves you are who you say you are.

The Authentication Service:

- receives authentication requests from principals,
- issues DCE identities (credentials) to users,
- provides mutual authentication between a client and a server,
- is based on Kerberos version 5 from M.I.T., and
- provides account passwords that are used for encryption and decryption and are not sent out over the network.

3.2.1 Session Keys

A session key is a temporary secret key that is generated by the Authentication Service. This key is made known only to the two parties requiring authentication. For instance, when the client sends a message to the server, the client would encrypt the message using the assigned session key. The server would then decrypt the message using its copy of the session key. If the message decrypts properly, the sender must be the client since the client is the only other principal that knows the key.

3.2.2 Tickets

Tickets provide the abilities to:

- protect the session keys from being tampered with;
- verify that the session key was generated by the Authentication Service; and
- prohibit anyone else from getting the session key even if they were to intercept the message that carried the session key.

3.2.3 TGT and PTGT

A successful DCE login leaves the user holding tickets or credentials. A ticket-granting ticket (TGT) granted by the Authentication Service is a user's basic proof of identity. A privilege ticket-granting ticket (PTGT) created by the Privilege Service is a list of security groups to which a user belongs.

3.2.4 Server Tickets

Clients need separate server tickets to talk to each server. The Security Service automatically constructs server tickets on request from authenticated users. Each server ticket is encrypted with the target server's key, clients don't know the key nor can they forge server tickets. A client sends the ticket to a server with requests. The server decrypts the ticket and responds.

3.2.5 Mutual Authentication

The exchange of tickets between clients and server securely gets a key to the client and server and provides mutual authentication between processes. The server believes the user's identity because the client holds a server ticket. The client believes the server's identity because the server can decrypt the ticket.

3.3 Privilege Service and ACLs

The Privilege Service creates privilege ticket-granting tickets (PTGTs).

3.3.1 Access Control Lists (ACLs)

Access-control lists permit applications to protect resources. They are similar to the UNIX filesystem's "mode bits", which divide the universe up into:

- Three categories of "who" are:
  - the owner of the file,
  - the group (i.e., other members of the owner's group), and
  - the rest of the world.
- Three categories of "what" are:
  - read,
  - write, and
  - execute.

When making a request to a server, a client presents its "certificate of identity", granted by the Security Service. The server:

- extracts the client's identity information;
- evaluates the combination of the:
  - client's identity,
  - resource the client wishes to access, and
  - the operation the client wishes to perform; and
- grants or refuses access if the client:

- has the necessary rights on the resource, the server authorizes the client and permits access, or
- doesn't have the necessary rights, the server refuses access.

3.3.2 ACL Managers

Servers implement access control in code called the ACL managers.

DCE does not use a common ACL manager. Each server that uses ACLs must manipulate its own ACLs. DCE applications don't even have to use ACLs; they can do authorization in any way they choose. Each application is free to manipulate ACLs as it sees fit.

3.3.3 ACL Inheritance

ACLs on the Registry allow access restrictions to propagate down the tree. Each directory has three ACLs:

- the actual ACL,
- the initial-container ACL, and
- the initial-object ACL.

A new directory in the branch inherits its parent's initial-container ACL. A new Registry object in the branch inherits its parent's initial-object ACL.

3.3.4 Keytab Files

Processes need a way to remember their DCE passwords. A keytab file is a local disk file that holds some principal's DCE key. A process uses a keytab file when acting as a client to automatically authenticate as some principal. A process uses a keytab file when acting as a server to decrypt incoming server tickets.

Access to keytab files is physically managed by the local operating system. Only authorized local users are granted read and write permissions. Keytab files are not stored in a distributed filesystem.

4. Time Service

DCE's Distributed Time Service (DTS) synchronizes time across all DCE machines and provides an API for applications needing accurate time information. Kerberos authentication requires that there is less than a five minute time difference between client and server.

The goal of DTS is to keep all clocks within a specified tolerance by reference to an outside, reliable time source and negotiating between machines to converge on the external time.

The two main problems DTS corrects are:

- Drift - the tendency of a clock to gradually deviate from the actual time.
- Skew - the difference between two clocks' values of the current time.

The Time Service includes DTS servers that maintain an accurate notion of the current time, adjust the clocks on their host machines, and provide time to DTS client processes. And, DTS clerks (clients) that receive time values from DTS servers and adjust their clocks based on input from DTS servers.

4.1 DTS Servers

Each server maintains a local representation of the time by:

- trying to contact a reliable outside source for accuracy;
- if not available:
  - ask all other servers for the time,
  - include its own time, and
  - compute a new time based on the inputs; and
- adjust its system clock accordingly.

4.2 DTS Clerks

Clerks request the time from a number of servers. The minimum number of servers is configurable. From these inputs, each clerk:

- computes the time and
- adjusts its system clock accordingly.

Clerks do not use their system clocks as an input. They attempt to use "close" servers (those on the local LAN).

4.3 Access Control

DTS maintains an ACL on each node's DTS process to govern the rights to manipulate DTS on the machine. Members of the DTS administration group automatically get all rights.

5. Host Services

Each host maintains local information about its DCE cell. This host-data information includes:

- the name of the DCE cell,
- any aliases for the DCE cell,
- the name of the host machine, and
- other information.

Administrators can define new host-data items for their own needs. Access to host data is governed by ACLs.

6. Audit Service

The Audit Service lets DCE record events by providing ways to:

- capture potentially critical events; and
- select the important events, based on
  - individuals causing events,
  - events themselves, and
  - combinations of the two.

Appendix C - Open Horizon's Connection

Open Horizon, Inc. is a leading provider of connectivity software that assists organizations in moving from departmental to enterprise-wide client/server solutions. The company builds and markets Connection, the first product to provide new or existing applications with plug-and-play access into enterprise services, such as heterogeneous databases, user authentication, data encryption, directory services and transaction processing monitors, for both two-tier and three-tier client/server architectures.

Open Horizon's Connection is a plug-and-play replacement for proprietary middleware products available from the relational database management system vendors. It provides a connectivity infrastructure that ties together disparate hardware, operating systems and networks into a single, integrated environment - a single computer. It provides transparent access to enterprise services, including database, security, directory, application and management services. Connection also allows an organization to simply plug in new or existing applications.

1. Application Broker

The Connection Application Broker (the "Application Broker") has been designed to meet the following organizational requirements:

- To enable any client application with the ability to transparently access virtually any remote business logic that has been implemented with a 3GL (such as C, C++, or COBOL), a 4GL (such as Dynasty), transaction processing monitors (such as Encina, CICS, Tuxedo, or Top End), legacy applications, or CORBA-compliant distributed business objects.
- To allow an organization to continue to leverage their investments in two-tier applications by providing a simple migration path to extend these applications to three-tier architectures.
- To support decentralized development teams in which the "front-end" GUI developers can work independently from the "back-end" business rule developers. While this separation is important, it can only be effective if both groups are capable of easily integrating their work.
- To be capable of using built-in functionality from leading GUI vendors' tools - such as PowerSoft PowerBuilder, Gupta SQLWindows, Borland Delphi Client/Server, and Microsoft's Visual Basic, Excel, and Word - to invoke remote business logic transparently.
- To support a constantly changing business environment, in which new business rules can be added and existing business rules changed dynamically at runtime, without the requirement to redevelop, recompile, relink and redeploy applications across the enterprise each time a change is made.

The Application Broker does not itself provide support for building the front-end GUI or the back-end business rules. Rather, it provides an infrastructure that allows front-end tools to access business rules that have been registered within it.

2. Architecture

Connection's architecture comprises two primary components, Connection Client and Connection Server. Connection Client resides on every client workstation in the environment, as well as every server platform that in turn must interoperate with other server platforms. A Connection Server library resides on every server platform that will participate in the environment.

3. DCE Compliant

Connection is fully DCE-compliant. Open Horizon is a member of the Open Software Foundation and chairs the DCE Database Special Interest Group.

4. Remote Procedure Calls

Connection utilizes remote procedure calls (RPCs) to communicate between each instance of Connection Client and Connection Server. Connection leverages the DCE RPC, but does not require the implementation of DCE Runtime or any other DCE service. All required base software comes bundled with the product.

5. ODBC Compliant

Connection Client with the ODBC Interface is itself an ODBC driver. When a client application issues an ODBC call, it passes the call directly to the Connection ODBC driver. Rather than convert the ODBC call into the target database API on the client, Connection takes the call as is and transports it across the network to the Connection Server that resides next to the target database. The Connection Server hands the ODBC call off to a server-resident ODBC driver, which converts the ODBC call into the target database API, and communicates directly with the database. Result sets are then sent back across the network to the client application.

6. Database Vendor API Support

In addition to ODBC support, Connection OCI Interface supports the Oracle Call Interface for accessing Oracle databases and Connection CT-Lib Interface supports Sybase's Client Library for accessing Sybase databases.

7. Three-Tiered Support

Connection supports a three-tier application architecture with client application access to business logic that is packaged for deployment to the second-tier application server. Connection transports client requests to the server in RPC format. On the application server, Connection formats the RPC into a C language function call. This C function call can be used to execute business rules that have been deployed in any of several formats: relational database stored procedures, C functions, etc.

References

[1] [Acly, 1996] Acly, Ed, *Connection from Open Horizon - Extending Data Access APIs to Support Enterprise Middleware Requirements*, International Data Corporation, January 1996.

[2] [Adams, 1996] C. Adams, *Independent Data Unit Protection Generic Security Service Application Program Interface*, Internet Draft, Bell-Northern Research, February 1996.

[3] [Chapman, 1995] D. Brent Chapman and Elizabeth D. Zwicky, *Building Internet Firewalls*, O'Reilly and Associates, 1995.

[4] [CIAC A, 1996] U.S. Department of Energy, Computer Incident Advisory Capability World Wide Web page, http://ciac.llnl.gov/ciac/SecurityTools.html.

[5] [CIAC B, 1996] U.S. Department of Energy, Computer Incident Advisory Capability World Wide Web page, http://ciac.llnl.gov/.

[6] [Cheswick, 1994] William R. Cheswick and Steven M. Bellovin, *Firewalls and Internet Security*, Addison-Wesley, 1994.

[7] [Colton, 1994] Colton, Malcolm, *SYBASE Secure SQL Server For Practical Multi-Level Database Applications*, Sybase, Inc., Technical Paper Series. 1994.

[8] [Colton, 1993] Colton, Malcolm, *SYBASE System 10, The Foundation for Enterprise Client/Server Computing*, Sybase, Inc., Technical Paper Series. 1993.

[9] [Cygnus, 1996] Cygnus Support World Wide Web page, http://www.cygnus.com/data-dir.html.

[10] [DOS, 1989] Department of State, *International Traffic in Arms Regulations (ITAR), 22 CFR 120-130, Office of Munitions Control*, November 1989.

[11] [DOS, 1992] Department of State, *Defense Trade Regulations, 22 CFR 120-130, Office of Defense Trade Controls*, May 1992.

[12] [Entrust] Entrust Home Page, http://www.nortel.com/entrust/.

[13] [Garbus, 1995] Garbus, Jeff, Solomon, David, and Tretter, Brian, *Sybase DBA Survival Guide*, SAMS Publishing. 1995.

[14] [Garfinkel, 1996] Simson Garfinkel and Gene Spafford, *Practical UNIX & Security*, O'Reilly and Associates, 1996.

[15] [GreatCircle] World Wide Web page, http://www.greatcircle.com/firewalls/vendors.html

[16] [Hu, 1995] Hu, Wei, *DCE Security Programming*, O'Reilly & Associates, Inc., 1995.

[17] [ITU-T, 1993] *ITU-T, Recommendation X.509, The Directory -- Authentication Framework*, International Telecommunications Union, Telecommunications Standardization Sector, Geneva, 1993.

[18] [Jaspan, 1995] B. Jaspan, *Kerberos Users Frequently Asked Questions*, Open Vision Technologies (http://www.ov.com/misc/krb-faq.html), September 1995.

[19] [McCurley, 1993] Kevin McCurley, *FY93 Technology Transfer Initiative Proposal: Information Integrity and Privacy for Computerized Medical Patient Records*, Sandia National Laboratories, 1993.

[20] [Merckling, 1994] R. Merckling and A. Anderson, *OSF RFCs on Smart cards and DCE*, Open Software Foundation DCE SIG, Request for Comments: 57.0, March 1994.

[21] [MicroSoft A, 1996] Microsoft BackOffice World Wide Web page, http://www.microsoft.com/backoffice/reading/bst11120.htm.

[22] [MicroSoft B, 1996] Microsoft BackOffice World Wide Web page, http://www.microsoft.com/backoffice/reading/bst10000.htm.

[23] [Navy] Navy ftp site, ftp://ftp.nrl.navy.mil/pub/security/nrl-opie/.

[24] [Neuman, 1993] C. Neuman, *The Kerberos Network Authentication Service (V5)*, Network Working Group, Request for Comments: 1510, September 1993.

[25] [Neuman, 1994] C. Neuman, *Kerberos: An Authentication Service for Computer Networks*, USC/ISI Technical Report number ISI/RS-94-399, Institute of Electrical and Electronics Engineers, September 1994.

[26] [NISTa, 1993] National Institute of Standards and Technology, *NIST FIPS PUB 46-2, Data Encryption Standard*, U.S. Department of Commerce, December 1993.

[27] [NISTb, 1993] National Institute of Standards and Technology, *NIST FIPS PUB 180, Secure Hash Standard*, U.S. Department of Commerce, May 1993.

[28] [NIST, 1994] National Institute of Standards and Technology, *NIST FIPS PUB 186, Digital Signature Standard*, U.S. Department of Commerce, May 1994.

[29] [Oceania, 1996] Oceania Inc., *Security Functional Requirements V.2.2*, January 30, 1996.

[30] [OpenHorizon] Open Horizon, Inc. World Wide Web page, http://www.openhorizon.com.

[31] [OpenHorizon A, 1995] Open Horizon, Inc., *Client/Server Connectivity - Beyond the Department to the Enterprise*, White Paper, December, 1995.

[32] [OpenHorizon B, 1995] Open Horizon, Inc., *3-Tier Client/Server Applications*, White Paper, December, 1995.

[33] [Orfali, Harkey, Edwards, 1994] Orfali, Harkey, and Edwards, *Essential Client/Server Survival Guide*, International Thomson Publishing, 1994.

[34] [OSF, 1992] *Distributed Computing Environment, An Overview*, Open Software Foundation, January, 1992.

[35] [OSF, 1996] *DCE Frequently Asked Questions*, February, Open Software Foundation, 1996.

[36] [OSF/DCE, 1996] Open Software Foundation, Distributed Computing Environment World Wide Web page, http://www.osf.org/dce/.

[37] [Purba, 1994] Purba, Sanjiv, *Developing Client/Server Systems Using Sybase SQL Server System 10*, John Wiley & Sons, Inc. 1994.

[38] [Purdue] Purdue Education ftp site, ftp://coast.cs.purdue.edu/pub/tools/unix/tcp_wrappers.

[39] [Reed, 1995] Reed, Paul and Jackson, Steve, *Separation Anxiety. Database Programming & Design*, October, 1995.

[40] [Rivest, 1978] R.L. Rivest, A. Shamir, and L.M. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM*, v. 21, n. 2, February 1978, pp. 120-126.

[41] [Schneier, 1996] B. Schneier, Applied Cryptography, John Wiley and Sons, New York, NY, 1996.

[42] [SEI, 1996] Software Engineering Institute World Wide Web page, http://www.sei.cmu.edu/SEI/programs/cert/.

[43] [SNL, 1993] CRADA No. SC93/01183, Appendix A, Sandia National Laboratories, July 26, 1993.

[44] [Sybase A, 1994] *SyBooks, SQL Server and Open Client/Open Server Release 10.0* online documentation. Sybase, Inc., 1994.

[45] [Sybase B, 1994] *System 10 Fast Track to SQL Server*. Sybase, Inc., 1994.

[46] [Sybase C, 1994] *SYBASE SQL Server Technical Overview*. Sybase, Inc., 1994.

[47] [Sybase D, 1994] *Tools and Connectivity Troubleshooting Guide*. Sybase, Inc., 1994.

[48] [Sybase E, 1994] *Sybase SQL Server System Administration Guide*, Sybase, Inc., 1994.

[49] [Sybase F, 1994] *Sybase SQL Server Security Administration Guide*, Sybase, Inc., 1994.

[50] [Sybase G, 1994] SQL Server v10.0 Reference Manual Volume 1, Sybase, Inc., 1994.

[51] [Sybase H, 1994] Secure SQL Server Security Features User's Guide, Sybase, Inc., 1994.

[52] [Sybase I, 1994] TransArc Corp., *DCE Secure Core System Administration*, Sybase, Inc., 1996.

Glossary

3GL	Third Generation Language
4GL	Fourth Generation Language
ACL	Access Control List
alias	An assumed or additional name
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
ATM (1)	Asynchronous Transfer Mode
ATM (2)	Automated Teller Machine
BSD	Berkeley Software Distribution
CDS	Cell Directory Service
cell	In DCE, a self-sufficient environment for distributed computing.
CERT	Computer Emergency Response Team
CIAC	Computer Incident Advisory Capability
Clearinghouse	A collection of directories in DCE. A DCE cell can have multiple clearinghouses, each holding some subset of directories in the namespace.
COAST	Computer Operations, Audit, and Security Technology Laboratory at
CORBA	Common Object Request Broker Architecture
COTS	Commercial-Off-The-Shelf
CRADA	Cooperative Research and Development Agreement
CT-Lib	Sybase's client libraries.
data security	The means by which the SQL server restricts access to the server, restricts access to data, restricts operations that can be performed on data, and maintain an audit trail that keeps records of who entered the system and/or used any system resources.database owner

DBMS	Database Management System
dbo	Sybase database owner
DCE	Distributed Computing Environment
DES	Data Encryption Standard
DF	Dedicated File
DFS	Distributed File Service
DNS	Domain Name Service
DOS (1)	Department of State
DOS (2)	Disk Operating System
DSA	Digital Signature Algorithm
DTS	Distributed Time Service
EEPROM	Electrically Erasable Programmable Read-Only Memory
EF	Elementary File
EMR	Electronic Medical Record
FTP	File Transfer Protocol
GDA	Global Directory Agent
GDS	Global Directory Service
GUI	Graphical User Interface
I/O	Input/Output
IDUP-GSS-API	Independent Data Unit Protection Generic Security Service Application Program Interface
IK	Internal Key
IP	Internet Protocol
ISO	International Standards Organization

ISQL	In Sybase, Interactive SQL user interface for accessing and updating system and user databases, tables, indexes, users, groups, etc.
ITU-T	International Telecommunications Union, Telecommunication Standards
LAN	Local Area Network
login	A login is synonymous to an account. It is comprised of a login name and a password.
M.I.T.	Massachusetts Institute of Technology
master database	The Sybase system database that records all of the server-specific configuration information, including authorized users, devices, databases, system configuration settings, and remote servers.
MLS	Multi-Level Security
NFS	Network File Service
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
OCI	Oracle Call Interface
ODBC	Open Database Connect
OS	Operating System
OSF	Open Software Foundation
PIN	Personal Identification Number
POP	Post Office Protocol
PPP	Point-to-Point Protocol
PTGT	Privilege Ticket-Granting Ticket
RAM	Random Access Memory
RDBMS	Relational Database Management System
Replica	A physical copy of a DCE CDS directory. A writeable copy is called the master replica and a read-only copy is called a read-only replica.

Replication Unit	A DCE CDS directory
ROM	Read-Only Memory
RPC	Remote Procedure Call
RSA	Rivest, Shamir and Adleman Algorithm
SGI	Silicon Graphics, Inc.
SLIP	Serial Line Internet Protocol
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
surety	A balance between confidentiality, Integrity and availability; a balance between unauthorized used of information and assurance of authorized use
Tcl	Task Control Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TGT	Ticket-Granting Ticket
TIS	Trusted Information Systems
UDP/IP	User Datagram Protocol/Internet Protocol
UUID	Universal Unique Identifier

Index

Distribution

5	MS 0449	Joselyne Gallegos, 9415
2	MS 0449	Victoria Hamilton, 9415
2	MS 0451	Timothy Gaylor, 9417
2	MS 1109	Kevin McCurley, 9224
2	MS 0661	Timothy Meeks, 4821
1	MS 1109	Art Hale, 9224
1	MS 0449	Judy Moore, 9415
1	MS 0661	Michael Pendley, 4821
1	MS 0451	Michael Sjulin, 9417
1	MS 1380	David Larson, 4231
1	MS 9018	Central Technical Files, 8523-2
5	MS 0899	Technical Library, 4414
2	MS 0619	Review and Approval Desk, 12630 For DOE/OSTI