Department of Computer Science

Magnus Persson

A Comparative Study of Security Features in FreeBSD and OpenBSD

Master's Thesis

2006:02

A Comparative Study of Security Features in FreeBSD and OpenBSD

Magnus Persson

This thesis is submitted in partial fulfillment of the requirements for the Masters degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Magnus Persson

Approved, March 1, 2006

Opponent: Thijs Holleboom

Advisor: Simone Fischer-Hubner

Examiner: Donald F. Ross

Abstract

Security in operating systems is a highly topical subject nowadays as the Internet keeps expanding. The larger the Internet gets the more systems, with valuable information, get connected, which could be subjects of attacks. An operating system needs to protect its information from these attacks. Many servers are using UNIX based operating systems and the security in these systems is a widely discussed topic. This project is going to test and investigate the security in two of the most common UNIX distributions, both based on the Berkley Software Distribution (BSD). The selected systems are FreeBSD and OpenBSD. The Add-on called TrustedBSD/SEBSD for FreeBSD will also be a subject for this project. A comparison of the security features in the two systems was performed both theoretically and practically and this report reflects the results of these experiments and comparisons. A conclusion is that each system suits best in different environments with different needs. The selected distributions also have different level of security in specific areas. An introduction to security in operating systems on a general basis is provided before the actual comparison begins.

Contents

List of Figures

List of Tables

Chapter 1

Background

This chapter will explain the purpose of this project and present the selected systems that were used in the experiment and explain how the comparison was conducted. Also the structure for the thesis is presented.

1.1 Introduction

UNIX based operating systems are today widely used both by private users as well as by companies.[23] There are many threats against an operating system. Often the threats are from attackers that want to gain administrator access to the computer. In a UNIX operating system the administrator account which has full access rights to the system is called *root*, also known as super-user account. Gaining administrator access will make the attacker get full control over the attacked computer. This threat may come from both remote and local directions. To get administrator access is just one way of many to attack an operating system. For example daemons, i.e. applications that run in the background as services, with super-user permission, may have so called backdoors. Backdoors are flaws in the software that the developers have left there by purpose or by mistake. A backdoor is like an open hole in an application that may give and attacker administrator access or may let the attacker get hold of information from the system. There are many other attack methods for getting root access and there are a lot of different methods to prevent this from happening, but some systems do not have this protection by default. This project is going to compare the security features in two selected UNIX distributions and one security add-on for one of the systems. UNIX distributions will from now on be referred to as UNIX systems, which in this report mainly refer to the selected distributions. The purpose of this comparison is to analyze if the selected UNIX systems are secure in an installation with minimal configurations made.

1.2 About the Project

1.2.1 Selected Systems

There are many different UNIX systems today, but some of the most common systems for private users as well as for companies are the BSD systems. As BSD is a free alternative to other operating systems many private users finds them attractive. But new users may use the systems without really knowing much about them and are not having too much experience with security concerning operating systems. This is the main reason for investigating the security in these systems. Companies are often aiming for higher security in their systems compared to most private users. But security for private users is also very important with today's highly expanding networks and accessibility through Internet. A system that is not secure may be hijacked by an attacker and used to attack other systems. The hijacked system may as well act as an e-mail spamming host or as a branch for other illegal activities. Hence an unsecured private system may not be a major problem for the owner, but if the system is used by an attacker to obtain control over other systems, which may be company systems, it may be a problem for other people.

The BSD distributions are under active development and have many contributions from users. As the distributions are developed under open source license there could be more exploitable flaws discovered faster but at the same time the flaws could be found by attackers first, which would use them to gain access to a system. Also systems that are widely used are having a higher risk to be attacked. With this in mind this project is going to investigate which of the BSD systems selected below are most secure with as little configuration as possible, which for example means configuration of network interface and keyboard. The definition of default installation in this project is that it should be a regular installation of the system, with the minimum of configuration made. The applications and services that have been installed depend on what has been included in the default installation for each system. The selected operating systems that will be compared for security features are the two UNIX systems FreeBSD[1], OpenBSD[2] and the add-on for FreeBSD called TrustedBSD[3]. Figure 1.1 show the BSD family hierarchy. Even though the TrustedBSD distribution is an add-on for FreeBSD it will in this project be treated as an own system in some parts, because of its major difference in security. Another reason for these selected UNIX systems is that FreeBSD and OpenBSD are both siblings from the same system, NetBSD. TrustedBSD may be seen as a sibling from both FreeBSD as well as from SELinux[26]. The base of TrustedBSD is FreeBSD and the add-on parts are from SELinux.

A comparison between these three systems will hopefully show which of them are most secure by default and perhaps what system to use if high security is needed. The developers of OpenBSD have a main goal of very strict security. FreeBSD is aiming more to user-friendly system administration and stability but of course to security as well. The study will show which of these systems to prefer depending on different situations. For a more detailed description of the selected operating systems see chapter 5.

1.2.2 Security Aspects

The definition of security in a system includes three different aspects, which is system availability, data integrity and data confidentiality. These three aspects are the backbone

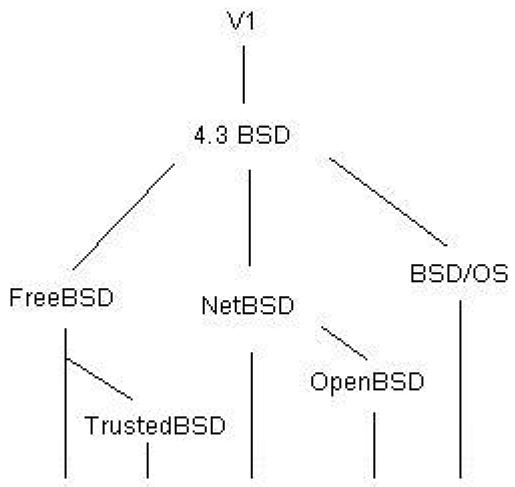


Figure 1.1: The BSD hierarchy.

in operating system security. System availability is whether the system and resources are available for use and should only be available for the intended users. Downtime of the system may be very costly and perhaps stop users to be able to do their work. An attacker may not break into the system, but the attacker may stop users to login or access a service, which could be just as bad as downtime or a system crash. Data integrity is whether the information on the system is unmodified. The user should be able to trust that the information is correct and has not been altered in an unauthorized manner. The last aspect is data confidentiality where the information on the system should be protected against unauthorized people. This is a measure of the ability of the system to protect the secrecy of its data.[35]

1.2.3 Comparison Method

The comparison in the project was performed theoretically and practically. The theoretical part consists of literature studies for the systems and the different areas of security. The practical part was based on experiments to the selected systems. The method of comparison was consisting of several sub comparisons of different security aspects. Some tests will be done to the different systems with vulnerability scanners to see what system has highest security by default according to known security issues. The details of the tests will be presented in chapter 6. The main security features that will be compared are Access Control, Authentication, Auditing and Logging, Encryption and some network security features including firewall installations. Also a general comparison will be done to the systems to see if there exists any major difference that may impact the results. The reason for the choice of security features is based on the most common and most important security areas that may exist on operating and networked systems. Access control manages the access of subjects (users, processes) to objects (files, directories, IPC, processes). Authentication manages the access to the system itself in different ways that is verifying the identity of a user. Logging is a way to collect information about what is happening or what has happened in the system and for different services that are running on the system. These logs may contain information about possible attacks or security breaches. Encryption is used for encryption of data over network, on the hard disk drive or other devices. Network security is about how to protect against remote threats like DoS (Denial of Service), how to deny specific IP addresses from accessing the system and how to transfer data secure over the network. A detailed description of the security features can be found in chapter 2.

1.3 Structure of the Thesis

Chapter 2 describes the design and structure of a secure operating system. It also describes a number of different security aspects to a system; Access Control, Authentication, Auditing and Logging, Cryptography, Firewalls and Intrusion Detection Systems as well as some cryptographically based network protocols. An introduction to the Common Criteria[33] for IT evaluation is also described in this chapter. Chapter 3 describes the security enhanced UNIX and Linux[30] add-ons that are discussed in this project. The chapter 4 discusses possible attacks to an operating system and gives an introduction how to secure a system against these attacks. Vulnerability analysis tools are used in this project to test the selected systems for exploitable bugs or weaknesses. These tools are described in a general introduction and then each selected tool that was used during this project is described. Chapter 5 is the chapter that compares the systems in the different areas described in chapter 2. It first gives a brief introduction to the selected UNIX systems and then the comparison is discussed for each area. A discussion for the comparison is given at the end of this chapter. This chapter is more of a theoretically comparison than the following experiment chapter 6, which is more practical. The experiment chapter is about the tests that were performed. Information about the experiment setup and the experiment results are given in this chapter. The last chapter, chapter 7, is discussing the project results and the authors point of view. Besides, it gives a short listing about what could be further tested to perform a in depth security analysis of the operating systems.

Chapter 2

Operating Systems and Network Security

This chapter explains each area of operating systems and network security in detail. The topics that are explained are the ones that will be compared for the selected operating systems. An introduction to how secure operating systems are structured and a brief introduction to the Common Criteria for IT-security evaluation is given.

2.1 Design Principles and Features of Secure Operating System

A secure operating system should not only be implemented with security in mind, it should also be checked for the security it provides after the implementation. This makes it hard to create a secure operating system from an already existing operating system that was not designed with this security from scratch. The TrustedBSD project made its own framework for the FreeBSD kernel (MAC-Framework) that was implemented after the FreeBSD was developed, which made it a hard task to create a secure operating system based on an already existing one. There are several design principles that make a secure operating system more secure than a regular operating system. A regular operating system could be seen as an operating system that has been developed for functionality with less security in mind. The design principles are for example the following[40]:

- Least privilege
- Economy of mechanisms
- Open Design
- Complete mediation
- Permission Based
- Separation of privilege
- Least common mechanism
- Easy of use

All users in a system should operate with the least privilege possible at all time. This prevents malicious software from gaining resources on the system and stops them from doing any harm. The design of the security in the system should be as simple as possible as the system should be easy to be tested and analysed and of course reliable (Economy of mechanisms). The protection mechanisms in the system should not ignore possible attacks and should be open for the public so the design of the security mechanisms can be confirmed by independent people (Open Design). All access to the system should be carefully checked and there should not be any way to go around the access paths in the system (Complete mediation). The system should be built up on permissions (Permission based) and as the default scheme for the system there should be denial of access for resources in the system. The objects that may be accessed should be identified. The access for these objects should preferably not rely on one condition but at least two conditions before access to the objects can be granted (Separation of privilege). As these objects may be shared between users or processes the information flow creates channels that may be attacked. Therefore the separation of logical and/or physical environment is needed to reduce these risks (Least common mechanism). Logical separation separates the objects of one user from those of another. Physical separation is when processes use different hardware facilities like separation between sensitive computation and non-sensitive computation. Temporal separation is also a choice where the sensitive and non-sensitive computation is computed in different time periods. A security mechanism should be easy to use, so that it is actually used, which is the last general design issue in a secure operating system. The figure 2.1 show the security mechanisms in a secure operating system with its secure access control for hardware and services.

The previous listed parts of a secure operating system were the design principles of a secure operating system. Security features such a system applies are [35]:

- User identification and authentication
- Access Control
- Object reuse protection
- Complete Meditation
- Audit
- Audit log reduction
- Trusted path
- Intrusion detection

A regular operating system usually has password authentication. In a secure operating system the authentication are performed with more secure methods that are explained in

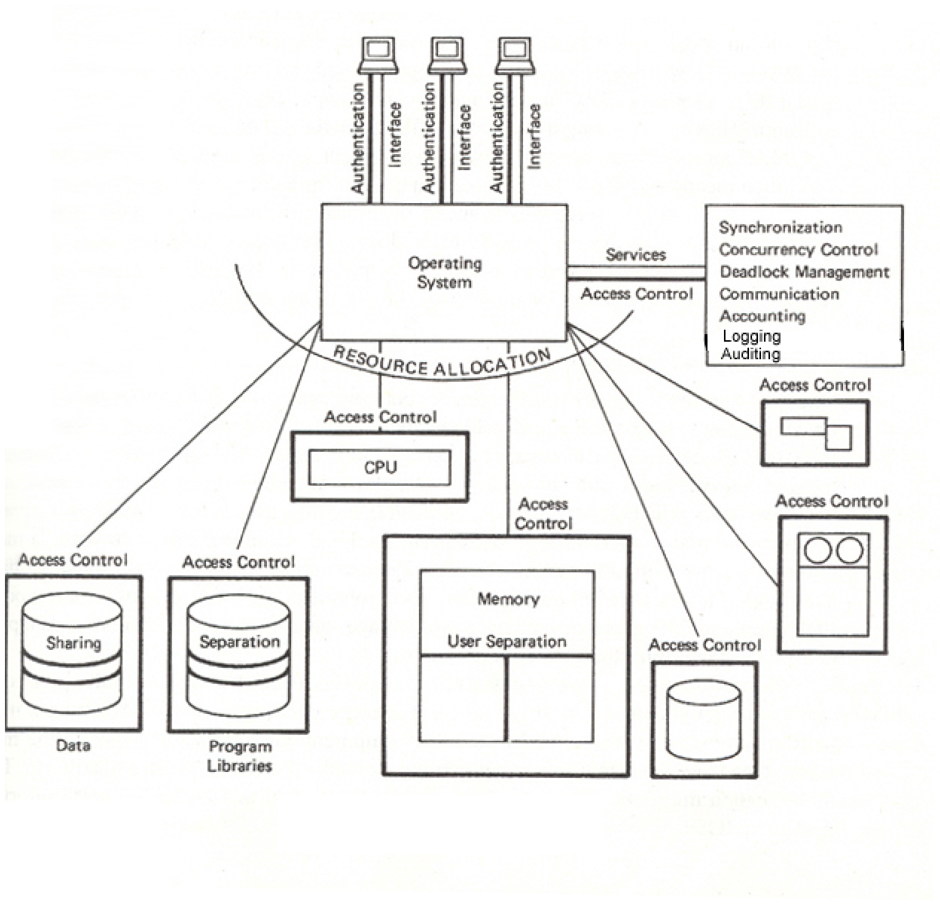


Figure 2.1: The security mechanisms for a secure operating system.[35]

chapter 2.3, Authentication. Another security issue in an operating system is to protect the memory from unauthorized access, i.e. Users reading other users/system data. This is usually done by virtual memory management or segmentation on hardware level. Access control is usually used to handle the access of subjects to objects such as directories, I/O devices, files and to prevent unauthorized access. Objects in an operating system needs to be protected as it may be shared objects between users and between processes. The allocation and access of objects can be done by lookup tables and the system needs to control the integrity and consistency of the objects. Scheduling algorithms make sure that users do not starve, that is running out of resources, while other users may hog resources. The

inter-process communication and synchronization is managed with access control tables for processes and users. All this protection and security attributes managed by a regular operating system is stored in databases. This data needs to be highly protected as if this data is changed the basic security in the system may fail to work. The protection of this security data is managed by hardware control, encryption and isolation that prevents the data to be deleted or modified.
A huge difference between a regular and a secure operating system is the separation and isolation of the security mechanisms. In a regular operating system the security kernel and operating system are used together, which creates security activities at several places in the system. This makes it harder to control all the security relevant issues and makes the system a lot more open for attacks, see figure 2.2 to get an overview of this issues. Figure 2.2 also illustrates the different places where security management is handled and there are many places where security breaches may arise.

In a secure operating system the security kernel is separated from the operating system activities and all security activities in the system is performed at one place in the system, in the security kernel. Having the security related issues handled at one single place in the system makes it a lot easier to control the system resources and services, it is also a lot more secure against attacks as it can be monitored and has only one access point. See figure 2.3 for an overview of the operating system with separated security kernel. The figure shows how the security is handled in just one place, the security kernel, instead of having security management in several different places in the design.[35]

There are even more strict specifications of an operating system than the *secure* operating system specification. A system with even stricter security specifications is called a *trusted* operating system. There are special demands for a secure operating system to be called a *trusted* operating system. All implementations of the security mechanisms need

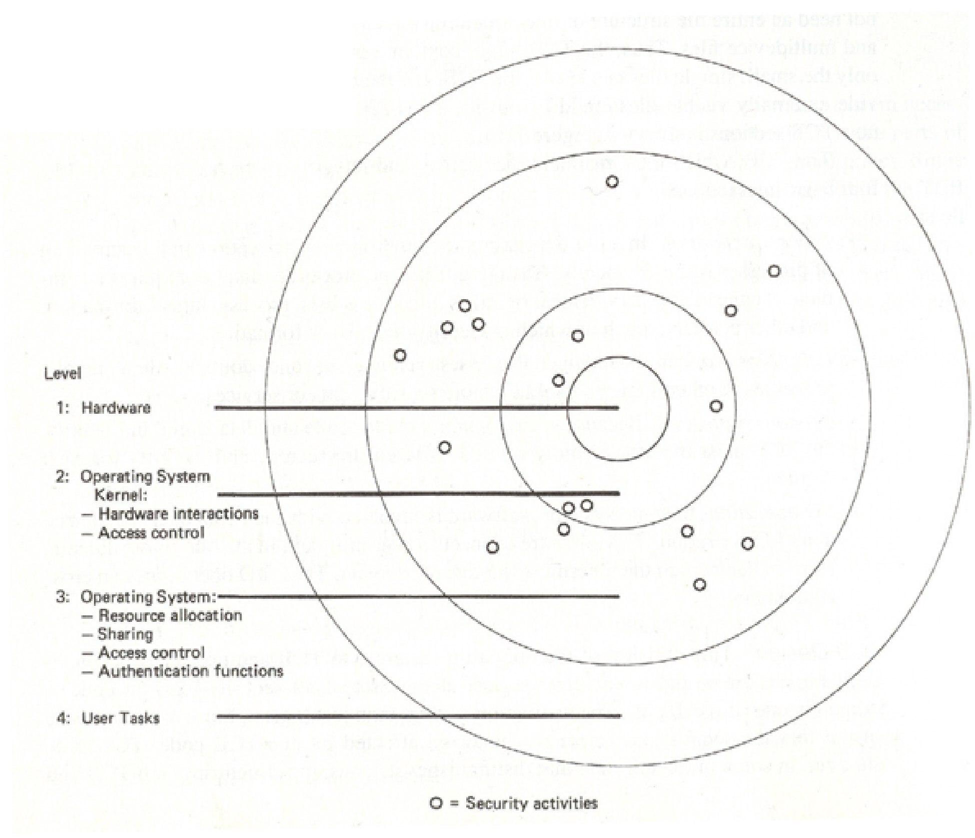


Figure 2.2: Security kernel and operating system intergrated. [35]

to be specified and verified. These verifications are very strict and there are a lot to verify before a secure operating system may be called trusted. This may be accomplished with the Common Criteria[33] by following the directives in Common Criteria and choose a level of evaluation that suites the specific project. The Common Criteria provide guidelines how to build a secure system and is explained in more detail in chapter 2.8.

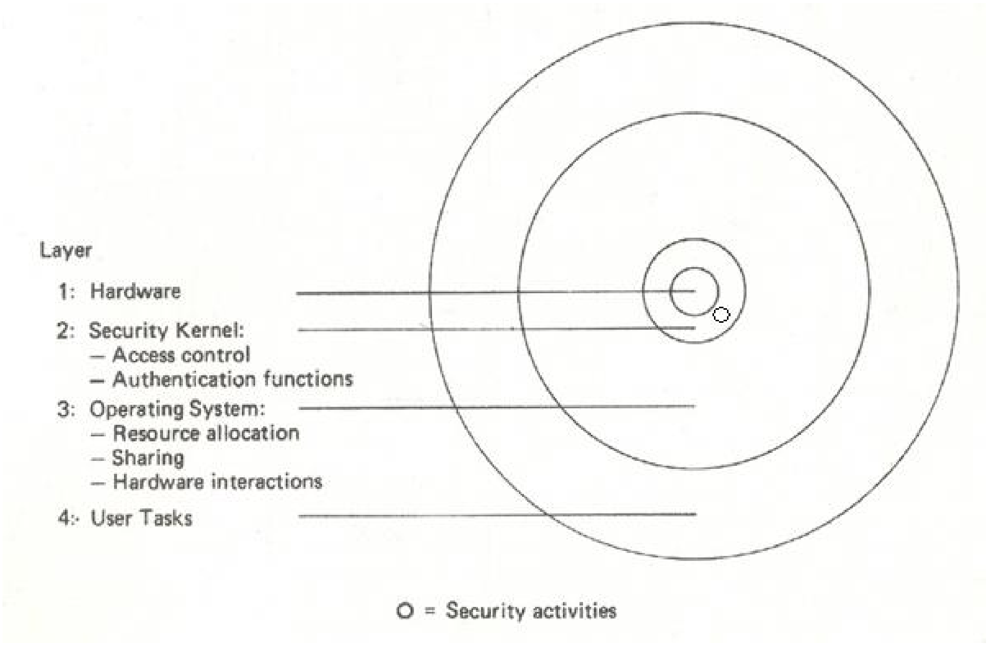


Figure 2.3: Security kernel and operating system separated. [35]

2.2 Access Control

Access control is not just access to the system itself (see chapter 2.3, "Authentication") but rather access to different objects, such as files in the file system. Not only files need to be controlled for access but also interprocess communication objects , devices, directories and other objects in a system. Many of these objects could be represented as files, for example in a UNIX operating system. Files have several types of attributes including;

- **Name** - Symbolic name for a file.
- **Identifier** - A unique tag for the file known to the file system.
- **Type** - The file suffix, depends if the file system support different types.
- **Location** - On which device a file is located.
- **Size** - The size of the file, usually in bytes.

- **Protection** - Access control information for the file.
- **Time, date, User identification** - Information about creation, modification and usage of the file.

The attributes that are important to access control are protection, time, date and user identification. The protection mechanism determines who can read, write and execute the specific files. The time and date at when a file was created, modified or used may be important for monitoring access to certain files. Monitoring of file access may be important in systems with high security. Operating systems can have seven basic file operations; append, create, write, read, repositioning, deleting and truncating of files. All of these seven operations need to be protected from users that should not have access to apply any of these operations. The most common approach to manage these access rights is by letting the access to the files depend on the identity of the user (known as discretionary access control, which is explained further in this chapter). Different users may need different access rights to same files or directory. The most general way to apply this user-identity access control is with Access Control Lists (ACL), which specifies the user name and access rights to each file or directory. If a user wants access to a file, the operating system searches for the access list associated with that specific file. If the user is included in the access control list for that file with the requested access, the user is allowed access. Otherwise the user is denied access to perform the specified operation to the file. The problem with access control lists is their length. With many users in the system the size of the access lists would be very long. The UNIX systems solution for this problem is by having three different classifications of subject groupings[43];

- **Owner** - The user that created the file is the owner of the file.
- **Group** - A set of users that needs the same access to the file.
- **Universe (world)** - All other users in the system.

Combining this access scheme with access control lists will give much smaller access lists for each file or directory as the access control lists do not have to contain all users if a file only has permissions for one group.

Capabilities is another way to control access rights to files or devices by giving a list to each user on the system that state what permissions the user has to each file on the system. Capabilities are more suitable for distributed systems but are still hard to give a good overview of the access rights for a given object. Another issue with capabilities is that it is hard to revoke a capability. A more lightweight protection scheme is the three field's protection used in UNIX. In this protection scheme there are only three fields that specify the access control for each file or directory. Each field are a collection of bits, which each prevent or allow access. Read, write and execute are the most common bits that can be set (rwx). *r* controls the read access, *w* controls the write access and *x* controls the execution access of the file. The three fields that contain these set of bits are for owner, group and universe (also called world in some operating systems). There are nine bits needed to have access control of the files or directory's with this scheme. This scheme is usually used in combination with access control lists in most UNIX systems. Some UNIX system has a + after the access rights (rwx) for the files if an access control list is present. The disadvantage with the three field's protection scheme is that it is not very fine-grained access control.[43]

Access control can also be controlled with security policy models. These can be grouped into two main classes [35]:

- **DAC** - Discretionary Access Control
- **MAC** - Mandatory Access Control

Access with DAC is based on the identity of the requestor (user) and access rules for the requestor. The access rights with MAC are based on a centralized control authority that sets the permissions. DAC does not distinguish between a user and a process executed by the user in the means of access control. This is mainly the problem with DAC because malicious software like Trojan horses may execute programs on the user's behalf that may gather information or abusing of the system. Even if every single access request is controlled against the authorizations information may leak through a process, as DAC does not enforce any control on the flow of information.

Access control with MAC enforces the security policy by having a centralized authority that set the rules. The most common MAC policy is *multilevel security policy* which is based on subjects and objects in a system. Objects are files, directories or devices while subjects are divided into two subcategories, users and processes. With this type of access control the system is more secure against malicious software but MAC is not totally secure, as there could exists so-called covert channels. This is an information channel that is not intended to perform a normal communication, which could be exploited to get hold of information. A programmer should never be able to get secret information from an application when it is deployed for usage. But a programmer can always find ways to get hold of this information from the application covertly. The programmer may alter the output for a application by changing only small parts of the output that would not be noticeable by the users. An example of this change could be the word 'TOTALS' which the programmer may output as 'TOTAL' and the users would not notice the difference that leaves 1-bit of information open for the programmer. This is one way of many how to create covert channels.

RBAC (Role Based Access Control) could either be based on DAC or MAC. RBAC is constructed with different roles that have specific permissions. RBAC is more used in organizations as it maps to the organizations structure. This policy handles more of the responsibilities that each user has than who the person really is as this is more important in an organizational view. Each role has its responsibilities and each role could consist of many different users. The roles are building up a hierarchy and users belong to the different roles.[36] There are other ways of access control for files and directory's. One way is to use passwords for each file. If the password is changed often and is randomly generated this could be an effective way of access control but there are some disadvantages of this protection mechanism. If there are many files that need protection, there will also be a lot of different passwords to remember for the users. If only one password is used for all the files, and this password is discovered, the access is granted to all of these protected files.[43]

2.3 Authentication

There are many different types of authentication systems when it comes to computers in general. The most common security issues deal with the access to the operating system itself.

In general the authentication may be performed with something the user knows (passwords), something the user is (biometric) or something a user has (smartcards). This is the identification of the user. The most common way to authenticate is by using a login name combined with a password. Once the identification has been made, the next step is to authenticate the user so it is actually the real user. This is actually a verification of the identification, which is traditionally accomplished with a password that matches the identification. The passwords should be properly formed to be secure, usually with a length of at least eight characters mixed with numbers. This password is kept secret from others and only the user should know it. These passwords are secure, if they are properly formed and if only the user knows it, usually the operating system only knows the encrypted form of the password.

But there are several ways to get hold of the password. One way is by a dictionary attack. Another way is through social engineering, which means that the attacker is trying to find out relevant information about the person and then for example trying to guess the password. If the attacker succeeds, access to the system is possible with the users account.

It is also important that the authentication process is secure with no plain-text passwords passed over network. Only encrypted connections are secure, as long as the attacker cannot decrypt the data that have been sent. Encrypted connections are protecting against packet sniffing, in the way that the attacker cannot read the password without decrypting the packets.

To get a more secure login system to the computer a biometric fingerprint scanner could be used to access the system. The advantages to use biometric fingerprint recognition are that users do not need to remember passwords and login names. Users that need to remember such details usually write it down, which implies a risk that intruders get hold of this information. However biometric fingerprint recognition is not totally secure and has disadvantages. They are very costly to implement because they need special equipment. Intruders can still get hold of a fake fingerprint of some person that has access to the system through the fingerprint scanner. This might be accomplished by collecting the persons fingerprint and reconstruct a fake fingerprint out of plastic that is added to the intruders own finger. Some people also think biometric scanning violates their privacy.

Having a biometric scanner before the usual login phase with passwords could be a good solution, but costly. When using both techniques it is harder for an intruder to get access to the system as the intruder needs both a fake fingerprint and the login information.

A more secure biometric scanner is an iris scanner that scans the human eye for unique patterns. This device is a lot more expensive than a fingerprint scanner and not as common.

Another way to secure the authentication for the system is to use smart cards. A smart card is a small plastic card with embedded microprocessor and memory. Smart cards could be used to login to the system both locally and remotely. There are applications that could be used together with smart cards to get access with *SSH*[48] to a system, one application that allows this is *SSH-smart*[12]. *SSH* is a protocol that provides encryption while connecting to remote computers, see chapter 2.7.2. There are also pluggable authentication modules (PAM)[27] that support smart cards on UNIX systems, which use

RSA[7] cryptography. Combining smart cards with biometric scanning is the best way of authentication but it is very expensive.

Another common way to authenticate users to a server is by using an authentication system named Kerberos[18]. This authentication system is used to eliminate the risk of an attacker pretending to be a authorized user of the system, or an attacker that alters the network address or perhaps is trying to get into the system by eavesdropping the connection and then is doing a replay attack. Kerberos provides a centralized authentication server that is used to authenticate users to servers or servers to users. The authentication system uses symmetric encryption (see chapter 2.5 *Cryptography*) for this purpose. The idea is that the server generates so-called tickets that the user receives when the user connects to the server for the first time in the session. The user than decrypts the ticket and sends it backs to the server to verify the identity of the user. Figure 2.13 shows an overview how the kerberos system works. The chapter 2.7.3 explains more in detail how the Kerberos system works.

There are also a lot of security issues concerning the authentication for system services like the File Transfer Protocol (FTP), which is a common way to share files between computers. Access to FTP server is usually performed with authentication such as a login name and password. Some systems have anonymous FTP as default, which is a potential security risk. A default configuration of the FTP server could make the anonymous login to have access to files that should not be allowed to be accessed through the FTP login or make anonymous users upload files to the computer. Another way of sharing files between computers is distributed file systems (DFS). DFS is more complex compared to FTP. The server in distributed file systems share files to clients that are specified on the server side. With this method there needs to be strong authentication and not only by IP which can be spoofed (imitated). This is usually solved with encrypted keys. In UNIX the distributed file system is called network file system (NFS). The authentication in NFS is accomplished through the client networking information by default. In this type of authentication the users ID must match on the client and server. If this is not the case the server would be unable to determine the access rights to the files.[43]

2.4 Auditing and Logging

After a system has been installed and configured it needs to be monitored for all the security relevant activism to detect and determine intruders on the system. This is a way to make sure that the security in the system is actually working. Monitoring the system will also make the administrator aware of any failures or abnormal behaviours in the system. Monitoring is also called auditing. In an operating system there are usually a lot of different log files that keep track of what has been going on in the system. In the first operating systems there were only some logging facilities to keep track on what users did and for who logged in and out in the system. In more recent operating systems there are a lot more logging performed. There are logs for what files have been transferred over the network, users that changed to super-user, hardware events, and almost all services on the system that runs as server are logging much of its activities. The logs are divided into operating system level, application level and network level of logging. On the operating system level there are logs about hardware behaviours, kernel messages and also login failures as well as succeeded logins. On the application level the applications that are running on the system collect information about different events and failures. There are usually different logs for each application. The network level is logging incoming and outgoing packets on the network interface.

Log files are forming an audit trail, which is a recorded history of events that has happened in the system. These log files are a good way to keep track of possible problems or attacks to the system. The logs are also a source of information to see what caused the failure and perhaps what to do to insure that it will not happen again. These detailed logs are always a subject for an attack. Deleting or alteration of the logs makes the attacker invisible for the system administrator. But there are ways to make it harder for the attacker to overcome the log files. To have strict access control to the log files is one way. Another way is to send the logs to another machine, so the system logs are always on a backup. The backup machine could be an even harder system to break through. The backup machine could also be connected directly to the log host and not over a network. [17] Figure 2.4 illustrates a log backup host that is not connected over any network but directly with a serial line, which would be a more secure way to perform backup of logs.

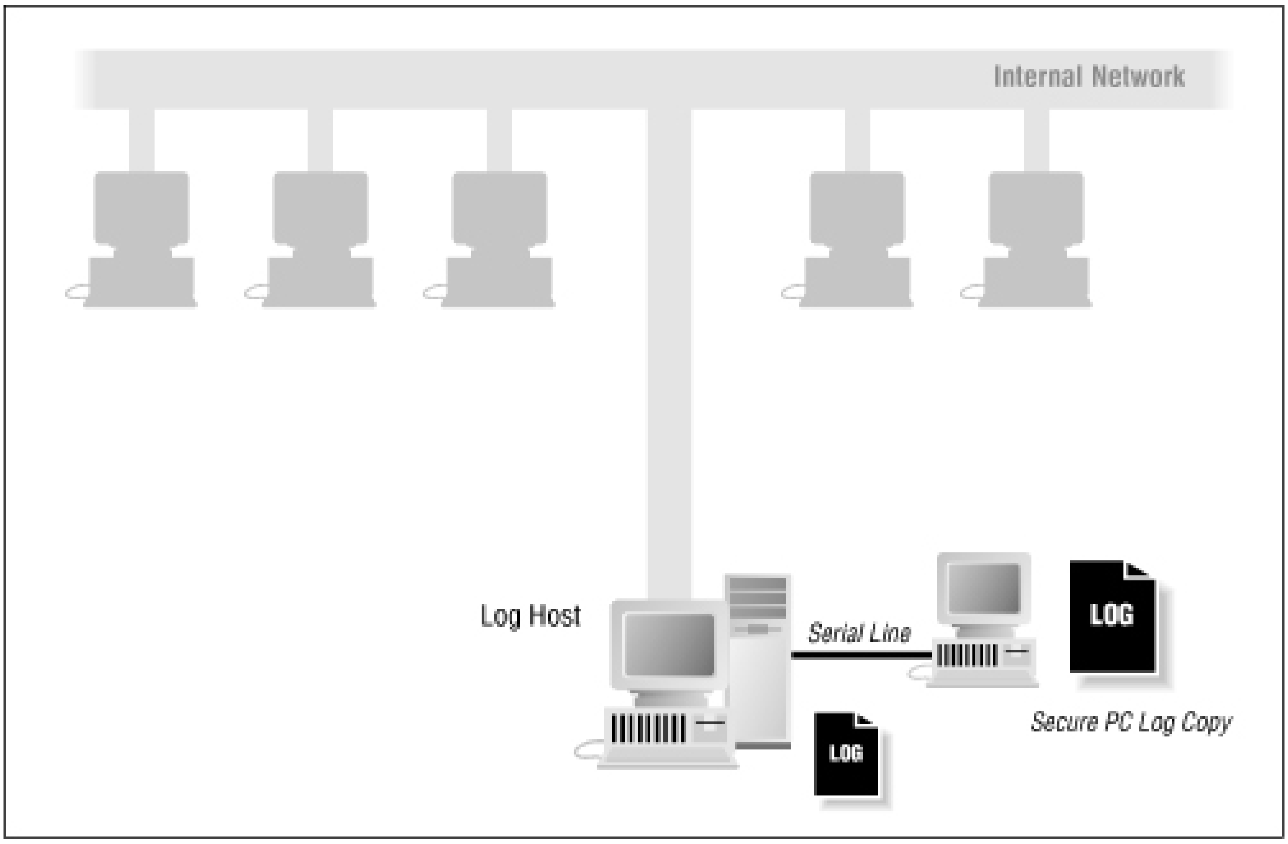


Figure 2.4: A backup of the logs may be stored on a another host. [17]

In UNIX there is a service called *syslog*, which was first developed for the Berkley sendmail[10] application by the University of California at Berkley software division. The system that uses *syslog* has a centralized system logging process running as a service.

Other applications that need to have information logged send the information to this *syslog* service. The log messages can be stored to files locally or on other computers depending of the sender and configuration of *syslog*. The information that is sent to the *syslog* service is the following; program name, facility, priority and log message. There is also the possibility to log over network with *syslog*, which is a good way to have backup of the logs.

Log watchers are good add-ons for the logging and auditing of the system. Log watchers allow the administrator to view logs in an effective way. There are also log watchers that let the administrator see the logs in real-time for active surveillance. These log watchers are usually shell scripts combined from different basic UNIX applications like *awk*[14], *sed*[15] and *wc*[16] but could also be more complex applications. An intrusion detection system is one example of a more complex application for this purpose, which also provides other features for system monitoring. Intrusion detection systems is described in more detail in chapter 2.6.2.

2.5 Cryptography

Cryptography is all about securing information. Encryption is where a message in plaintext is transformed into another message in cipher text using a mathematical function and a special password for the encryption process. This password is usually referred to as the *key* for the algorithm. Decryption is the opposite of encryption. The encrypted message is transformed from cipher text into plaintext through the mathematical function with the key. There are two types of encryption, symmetric and asymmetric. Symmetric encryption can be used to have a secure encrypted connection over networks, e.g. the Internet. Symmetric encryption is using the same key for encryption as for decryption. A symmetric encryption algorithm often uses Feistel (Horst Feistel, IBM 1973) cipher structure, confusion, diffusion and mode of operation. Diffusion means that a small change in the message gives a big change in the cipher text. With confusion, the key and cipher text should have as small relationship as possible. Feistel cipher structure is a method to encrypt a message with diffusion and confusion applied. The mode of operation may be an Electronic Code Book (ECB). This electronic codebook is always producing the same block of cipher code for the same given plain text. ECB is not very secure and Cipher Block Chaining (CBC) could be used instead. Figure 2.5 illustrates how CBC works. In the figure IV is the initialization vector, K is the key, P is the raw data and C is the encrypted data. The initialization vector for the next encryption is the previous output except for the first encryption. By using the previous initialization vector makes the encryption more secure as the initialization vector is different every stage in the process.

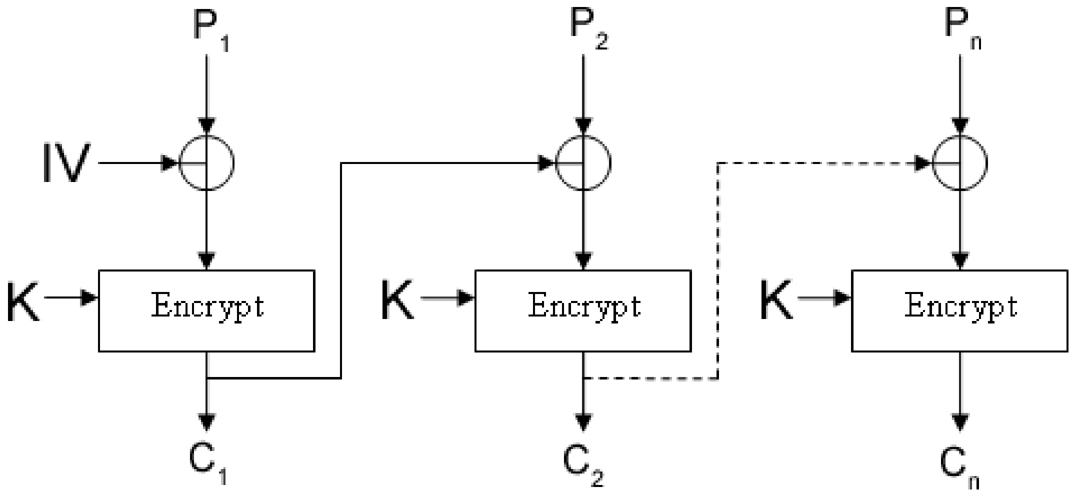


Figure 2.5: Illustration of Cipher Block Chaining.

Some algorithms that use symmetric encryption are DES[41], 3DES[41] and AES[11]. The last algorithm, AES, does not use Feistel cipher structure. 3DES is a lot better to use than the regular DES because it uses 168 bit (56*3 bit) keys instead of 56 bit keys, which DES does..

Asymmetric encryption is using two keys, one for encryption and the other one for decryption. Both keys can also switch places, at one time the first key can be used to encrypt, and at another time the same key can be used for decryption. The most used asymmetric algorithms are RSA (Rivest, Shamir, Aldeman) and Diffie-Hellman[13]. This encryption method is also known as public-key cryptography. The key to encrypt the message with is open for public and the key to decrypt the message is private.

There is a cryptography method that is called "One-way Hash", which is used for creating a fingerprint for a message. This is used to see if someone has changed the information in the message or to create a digital signature for the message that proves the identity of the author. The most common algorithms that are using this type of fingerprint encryption are MD5[41] and SHA-1[41]. The cryptographic hash functions have several properties for it to work, but two of the most important features are that inputs should not have the same output and a different input for a given input should not give the same output. Regarding hashed passwords, there is only one way to get hold of the hashed password and that is to brute force search the whole domain of possible passwords. That is searching all possible passwords to get hold of the corresponding hash value as the same plain text always generates the same hash value.

"Digital Signatures" is using the *one-way hash* method. One commonly used digital signature algorithm is DSS[19]. An authenticator is created by using hash-value of the message and is in turn encrypted by the sender's private key. The receiver decrypts the authenticator using the sender's public key. If the authenticators do not match, the message has been altered. The message could be sent in clear text. The problem with symmetric encryption is that the key need to be shipped to the receiver and cannot be encrypted with symmetric encryption itself. But combining both these encryption techniques, symmetric and asymmetric, will create a rather secure information flow. The symmetric key would be encrypted with the public asymmetric key. There are four desirable properties of cryptography, which are:

- **Confidentiality** - Only the authorized person or application should be able to decrypt an encrypted message.

- **Integrity** - It should be able to detect any type of alteration of the message during transfer from one source to another.
- **Authentication** - It should be able to identify the sender of the message, so it is really the supposed sender.
- **Non-repudiation** - The sender should not be able to deny that the message was sent.

These four properties are not always needed in all areas of cryptography. For example if the sender would like to be anonymous after he sent the message, non-repudiation is not desirable.

There are many types of usage for cryptography within an operating system. First there is usually some sort of encryption of passwords for the system and for other services. Second there is encryption of data over network and third there is encryption of hard disk drives. In all of these three cases, it is all about protecting data from unauthorized people. The only person that can access all files in a UNIX system is the administrator, i.e. root. But the administrator can be an attacker that has gained administrator privileges. This is where cryptography is good, even if the administrator is authorized to view all files in the system, the administrator cannot read the encrypted files even if he can access the files. The key is needed to decrypt the encrypted files and is not stored on the system. The algorithm that has been used is usually known and not kept secret.

Cryptography can also be used to detect alteration in data that may be accidental or intentional. By using digital signatures, the author of a document can be proven. Even if the data is encrypted there are ways for an attacker to get hold of the data. The attacker may try to brute force search for the used key. Depending on what the country allows in encryption techniques there are different algorithms used. United States has very strict laws that operating system need to obey when it comes to cryptography. The export of strong encryption is not legal in U.S. This is why some systems use other cryptography algorithms in their US versions that are not as strong as the newer algorithms used, for example AES. The information that is protected with encryption is protected with the strength of the key, the strength of the encryption algorithm and the specific implementation of the algorithm.[17]

2.6 Network Security

This section will discuss security regarding the network parts of the operating system, which includes firewalls and intrusion detection systems.

2.6.1 Firewalls

Firewalls are used to defend both systems and whole computer networks. The name firewall comes from the construction industry where they place special walls between sections in houses that are resistant to fire, which will slow the process of spreading of fire. The same idea is behind computer firewalls where the firewall is used to slow down or stop a possible attack. Firewalls also have other usages. Some of them are:

- Block access to specific Internet sites
- Block access to system services like ftpd or httpd
- Monitor network traffic between networks or the network traffic on a local machine
- Block specific IP addresses to access the machine

Even though firewalls are powerful tools, they should never be used instead of other security mechanisms. An attack may not come from a remote host but to the computer itself. The attack could also come from a trusted host, which the firewall lets through. Firewalls should be used in addition with all other security mechanisms that are used on a system. Firewalls are often used in company networks and other large networks to protect their local network from outside threats. But firewalls can also be used, and are often used, on the system itself. This is to protect the machine from attacks aimed for different services that the machine offers. Attacks are often targeting specific services like httpd or ftpd. If a firewall only lets through IP addresses that are specified by the system administrator the firewall will not accept incoming connections from other hosts, which are one way to improve the system security. Firewalls (usually personal firewalls) may also protect against worms and other malicious software in one aspect. This is by blocking malicious software from opening new ports, connect to remote hosts or let other remote hosts connects to the local host, which is something a worm usually do.

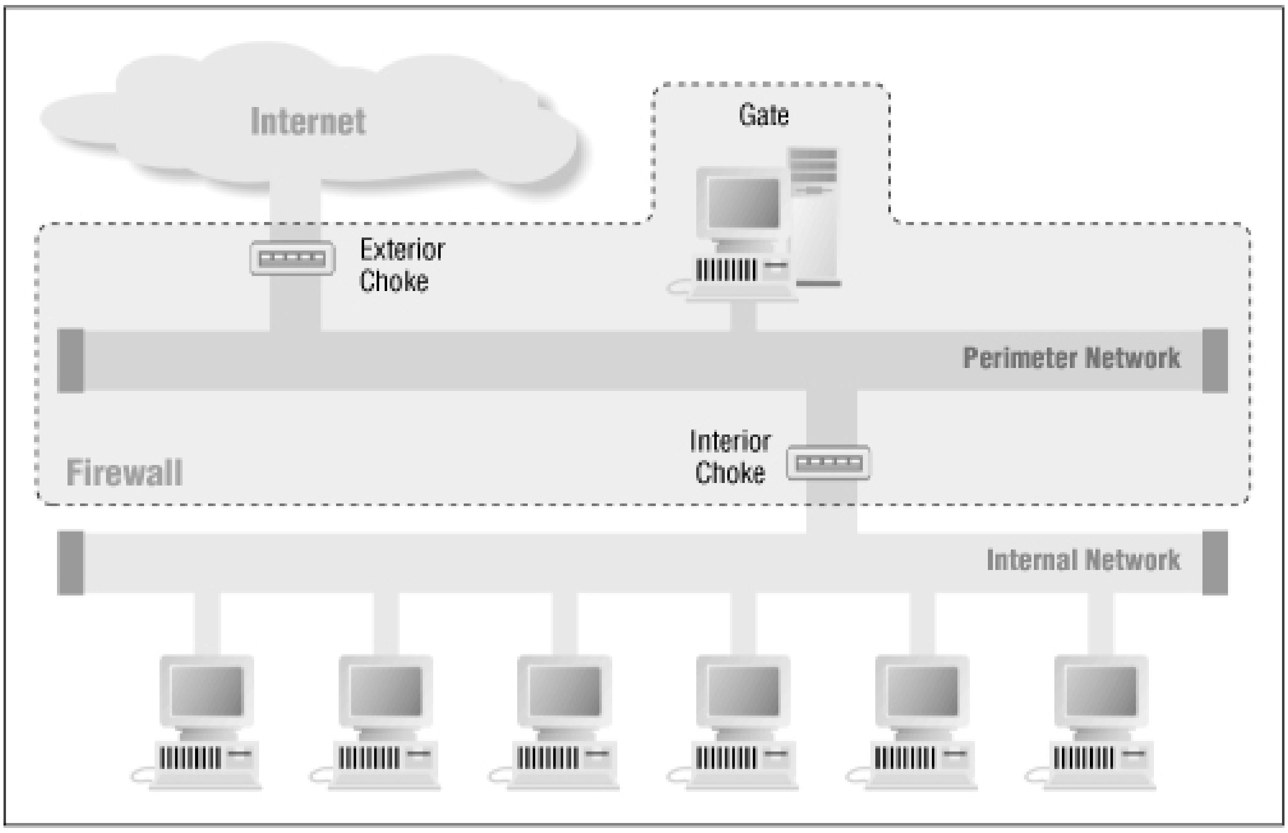


Figure 2.6: A firewall protects the internal network from an external network like the Internet.[17]

The main function of a firewall is to restrict the information flow between two networks, often local and global network, as mentioned above. Figure 2.6 illustrates how a firewall protects an internal network from an external, usually the Internet. The information control is controlled by different policies that the administrator set up. There are two basic ways of defining a policy for a firewall, which are default permit and default deny. With the first policy it is defined to let information that is not denied to pass through. The second strategy is to deny all information that is not specific defined to pass. Both these strategies have advantages and disadvantages. The "permit-all" strategy is easy to maintain but not that secure. The second strategy is hardening the security but may cause programs not to function correctly and users might need to use protocols that the policy deny. Combining these two strategies will create a good filter for packets streaming in and out on the system. There are different types of firewalls for networks and operating systems. There are chokes, gates and personal firewalls. Chokes are firewalls that might be a computer, a router or other communication device. The choke controls the flow of packets between networks. Gates are applications, devices or a computer that get incoming connections from external networks and forwards them by specific rule sets. Personal firewall is a system firewall that controls the flow of information in and out on a single computer by controlling the network interface. It acts as a gate but is only used for the system itself.

2.6.2 Intrusion Detection Systems

An intrusion detection system inspects all network traffic and identifies different specified patterns that may be a system or network attack. There are several different types of an IDS system. There is misuse detection where the IDS analyzes the information gathered from the network traffic and compares it to a database of attack signatures. The signatures in the database are for already known attacks. An IDS that use misuse detection is as good as the database is filled with signatures. Another type of IDS are anomaly detection where the system administrator defines the usual amount of network traffic, protocols, typical packet size etc, and the IDS look for anomalies from the specified settings. These definitions can also be based on statistics that the IDS itself gather from the network or system. These statistics are the regular amount of traffic and usual system behaviour.

An IDS can be network based or host based. An networked intrusion detection system is checking network traffic over the whole network and the host based IDS are only checking the system activities on the particular host. The IDS can also be passive or reactive. The passive IDS detects an attack or security breach and sends an alert to the system administrator. The reactive IDS will take actions to the attack like logging off a user from the system, or perhaps changing the firewall rules to stop the current attack. Changing firewall rules may also provide protection against further attacks.[46] One of the most commonly used intrusion detection systems at the moment is Snort[32].

2.7 Encryption Protocols

This section will discuss encrypted protocols and login systems like secure IP packets, kerberos and secure remote connections through SSH.

2.7.1 IPSec

IPSec[47] is a protocol that supports encryption of its data packets on the IP-layer. IPSec consists of three parts:

- **Authentication Header** - Authentication between hosts
- **Encapsulated Security Payload** - Encryption of the packets payload
- **Security Association** - Connection between two hosts over network like the Internet.

IPSec has some important features that the regular IP protocol lacks. IPSec supports access control, connectionless integrity, data origin authentication, and rejection of replayed packets, confidentiality and limited traffic flow confidentiality. The Security Association is a connection between two hosts that can offer security services to the traffic. Security Association can either be used for Authentication Header (AH, figure 2.7) or Encapsulated Security Payload (ESP, figure 2.8) but not both on the same Security Association.

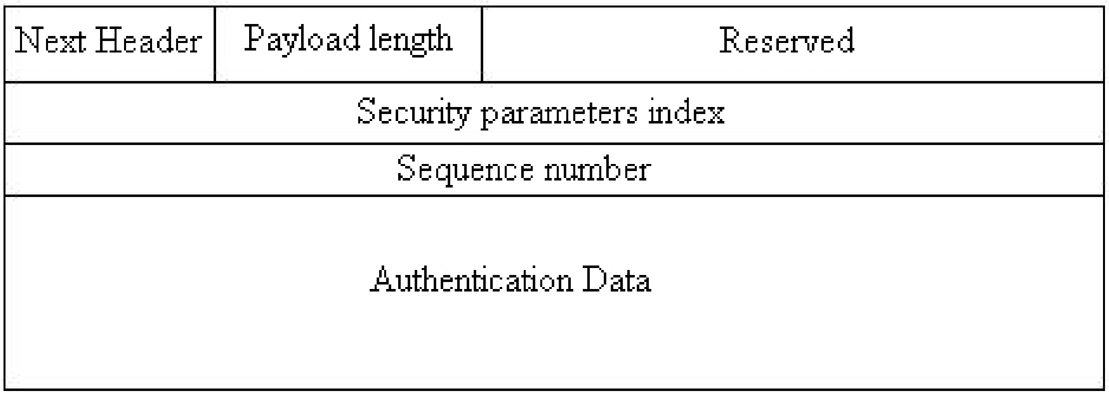


Figure 2.7: Authentication Header for IPSec.

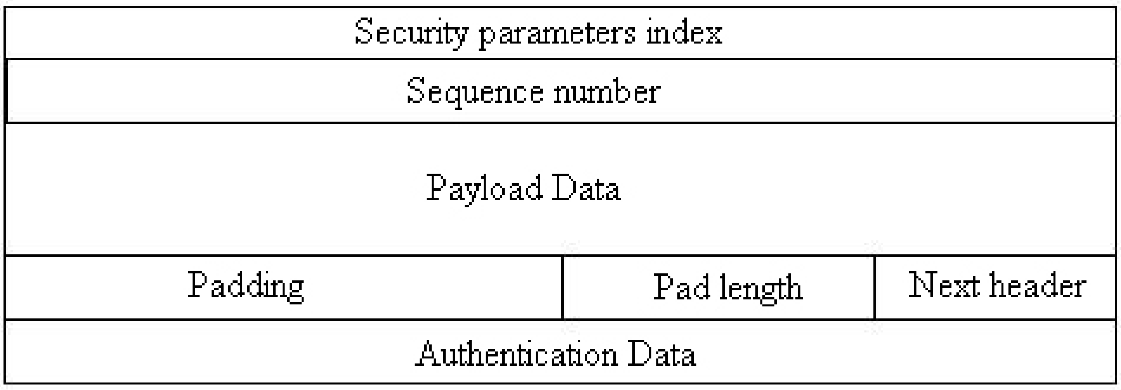


Figure 2.8: Encapsulated Security Payload for IPSec.

There are two types of Security Associations, which are Transport Mode and Tunnelling Mode. Transport mode creates an association between two hosts while Tunnelling mode lets the Security Association be applied to an IP tunnel. The figure 2.9 and 2.10 are illustrating how the Security Association is handled for IPv4 (IP version 4) in transport mode and tunnel mode for the use of Authentication Header. The same principle is applied for IPv6 (IP version 6).

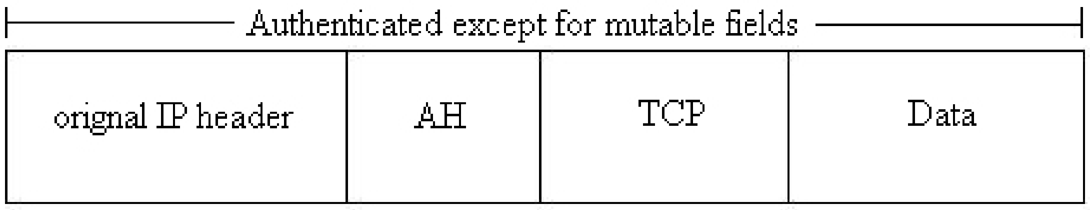


Figure 2.9: Security Association with the use of Authentication Header (Transport Mode).

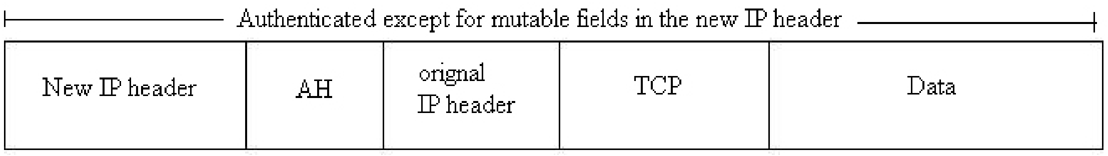

Authenticated except for mutable fields in the new IP header
New IP header
AH
orignal IP header
TCP
Data

Figure 2.10: SA with the use of Authentication Header (Tunnel Mode).

Another way to use IPSec is by using the Encrypted Security Payload functionality, which provides confidentiality services and encrypts the payload of the IP-packets. Figure 2.11 and 2.12 illustrates how this is accomplished over IP-packets for IPv4, which is the same way as IPv6 packets are treated.

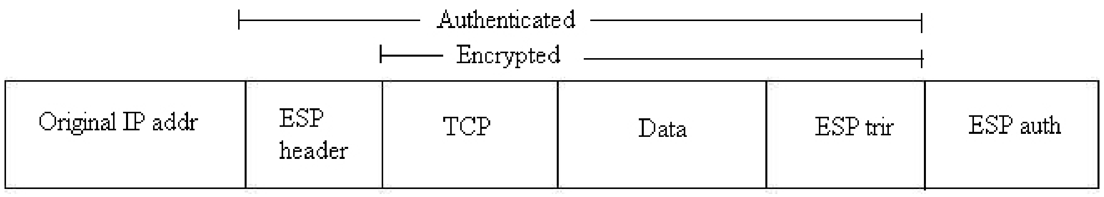


Figure 2.11: ESP with the use of encryption and authentication (Transport Mode)

Authentication Header and Encrypted Security Payload can be combined if several overlaying tunnels of Security Associations are used. IPSec is mandatory for IPv6 but could be used with IPv4 as well.

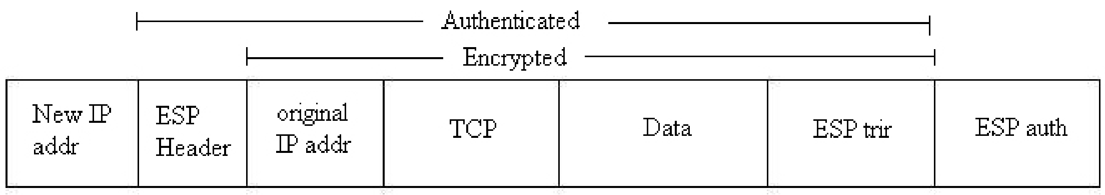


Figure 2.12: ESP with the use of encryption and authentication (Tunnel Mode)

2.7.2 SSH

Secure SHell is a protocol for securely connecting to remote hosts. It provides encrypted data transfer and let the user execute commands on the remote host. The SSH protocol is a replacement for the old *rsh*[39] and *rlogin*[38] which are not encrypted protocols. Connections to X-Windows[29] servers can also be forwarded over the secure connection, which means that the user can execute X-Windows based application on the remote host but view them on the local host through SSH. The user that connects to a SSH server needs to prove the identity before the login may succeed. How this is accomplished depends on which version of the protocol is being used. It is possible to just execute one command on the remote host if the *command option* is specified. This makes the server execute the command instead of starting a remote shell for the user.

The SSH protocol version 1 is based on RSA host authentication. This encryption method is based on public-key cryptography. The authentication method is secure against IP spoofing, domain name system (DNS) spoofing and routing spoofing. The user can automatically login if the user generates the RSA key with the key generator (ssh-keygen) and stores the key information files in the users home directory. SSH protocol version 2 has three ways of authentication for the user. First the client tries to authenticate using host based method, if this method fails the public-key method is used and at last the password authentication with user input is acquired. The password is sent encrypted to the remote host. The public-key method is using both RSA (as protocol version 1) and DSA[19] algorithm. The data sent between the hosts when using protocol version 2 is encrypted with AES, 3DES, Blowfish[41], CAST128 or Arcfour[31].

The differences between protocol version 1 and protocol version 2 are that the first protocol is lacking a strong mechanism for ensuring the integrity of the connection while the second protocol are using MD5[41], SHA1[41] and RIPEMD-160[41] for ensuring the integrity. The second protocol is supporting a wide range of encryption algorithms for the data transfer that the first protocol do not. [48]

2.7.3 Kerberos

One way to authenticate users to a server is by using a system named Kerberos[18]. This authentication system is used to eliminate the risk of an attacker pretending to be an authorized user of the system. It also eliminates the risk of an attacker that alters the network address or perhaps by trying to get into the system by eavesdropping the connection and then to do a replay attack. Kerberos provides a centralized authentication server that is used to authenticate users to servers or servers to users. The authentication system uses symmetric encryption (see chapter 2.5 *Cryptography*) for this purpose. To provide secure transferring of the keys for the symmetric encryption, asymmetric encryption can be used. The idea is that the server generates so-called tickets that the users receive when the users connect to the server for the first time in the session. The user then decrypts the ticket and sends it backs to the server to verify the identity of the user. Figure 2.13 illustrates how the Kerberos system works. First the user has to request (1) a ticket that the user needs to decrypt (2). The authentication ticket is sent back to the ticket granting service (3). If the user has access with a correct ticket the ticket granting service sends back a ticket to the network services (4). The user can then authenticate with this ticket to the network service (5) and a regular client/server session is established between the user and the network service. Another authentication system like Kerberos is X.509 [4] which is based on certificate authentication with public key cryptography.

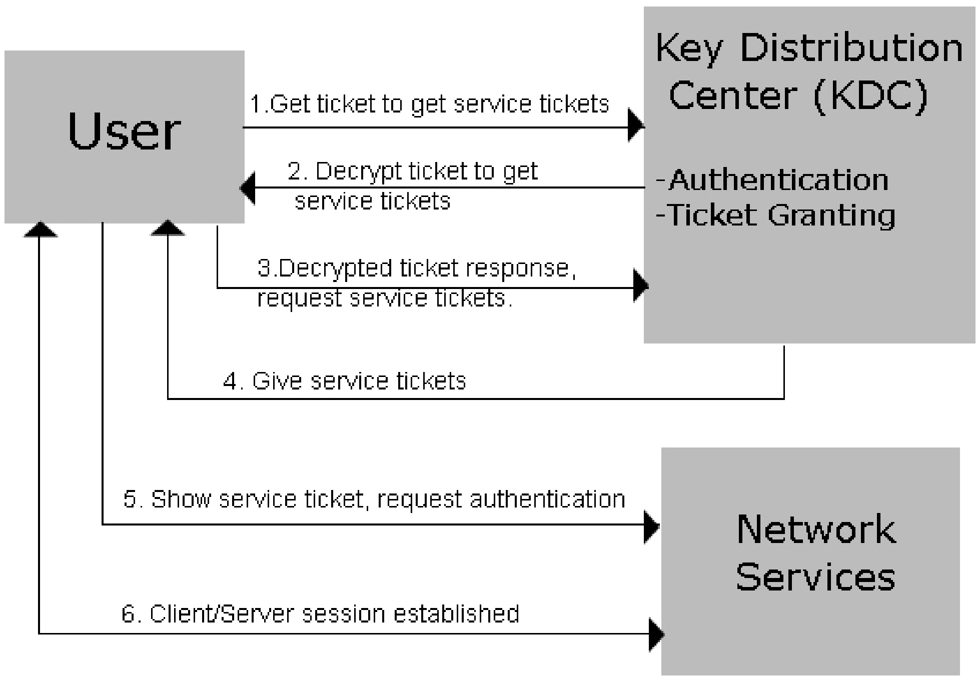

User
Key Distribution Center (KDC)
-Authentication
-Ticket Granting
1.Get ticket to get service tickets
2. Decrypt ticket to get service tickets
3.Decrypted ticket response, request service tickets.
4. Give service tickets
5. Show service ticket, request authentication
6. Client/Server session established
Network Services

Figure 2.13: Kerberos mechanism for user authentication and ticket passing.

2.8 Common Criteria

The Common Criteria is a criteria catalogue for evaluation of IT security. It is designed to be used by the international community to have compatible results of a security evaluation of IT products or systems done by different national evaluation parties. It also provides an objective yardstick for comparing the levels of security of different systems or products. There are three parts in the Common Criteria; Introduction and general model, security functional requirements and security assurance requirements, and all these are applied for consumers, developers and evaluators in different ways. The security criteria catalogue has five parts of concepts:

- **Security Environment** - Laws, Organisation policies etc.

- **Security Objectives** - Statements for the identified threats and satisfaction of organisational security policies.

- **Target of Evaluation Security Requirements** - Technical requirements
- **Target of Evaluation Security Specifications** - Define a implementation scheme for the target of evaluation.
- **Target of Evaluation implementation** - Realisation of the Target of Evaluation according to the specification.

The Target of Evaluation is the objective to secure including its environment. The Common Criteria are not aiming for only software development but also for evaluation of already existing systems and applications. When an evaluation is about to be performed usually a protection profile is identified. The protection profile is defining security requirements and objectives. The protection profile is implementation-independent and forms the profile for a certain category of products, which has the same needs for IT security. There has already been developed protection profile's for firewalls and databases etc. A security target contains IT security objectives, requirements and forms the basis of an evaluation based on the TOE. The security target may need several protection profile's if the target of evaluation consists of different categories. There are several areas for the evaluation that aim for specific environments, for example trusted paths, authentication, auditing, communications etc. The security assurance for the target of evaluation has seven different hierarchically structured security levels where each level explains what the evaluated system needs to fulfil to reach that specific level. These levels are called Evaluation Assurance Levels.

- **EAL1** - Minimal Protection
- **EAL2**- Discretionary Security Protection
- **EAL3**- Controlled Access Protection

- **EAL4**- Labeled Security Protection
- **EAL5**- Structured Protection
- **EAL6**- Security Domains
- **EAL7**- Verified Design

The first four levels of assurance are pretty basic and without specialised security engineering techniques. These levels are not too hard to reach for developers without too much cost involved. Though the forth level is quite advanced but everything is relative depending on resources and economy. The next three levels, five to seven, are much more complex and are reached if the target of evaluation was designed with strict requirements from the scratch. These systems are usually military systems and trusted systems. The evaluation of a system is specified by following the advices in the Common Criteria and see if the system follows the evaluation assurance levels and requirement specifications. The protection profile, security target and target of evaluation are evaluated to see if they correspond to the defined criteria.[33]

2.9 Summary

This chapter has described the basics in computer security regarding operating systems and networks. The parts described are the ones that are about to be compared in the chapter Comparison of Security (chapter 5) and also some parts in the vulnerability experiment chapter (chapter 6). Access Control, authentication, auditing, logging and cryptography were briefly described as were the security issues related to these topics. The network security section describes firewalls and intrusion detection systems that could be used to prevent or detect attacks. The most common secure protocols were described, which could be used in the tested operating systems. The cryptographically based protocols that were discussed were; IPSec that is used for the IP layer, SSH that is the most common encrypted remote login protocol and Kerberos, which is a common authentication system. The last section in this chapter describes Common Criteria, which is a common framework for evaluating a system or software. This is discussed as it could have been in theory used for this project but is way too massive to use in this particular case. The main goal for this chapter was to give an introduction to computer security in general and at the same time give relevant information about what is going to be compared between the operating systems.

Chapter 3

Security Enhanced UNIX Systems

The following chapter explains the SELinux/Flask and TrustedBSD/SEBSD security architectures. Also a Linux specific security architecture called RSBAC is presented.

3.1 SELinux and Flask

The National Security Agency[5] (NSA) started to develop the SELinux distribution from the Flask[22] architecture, which had been developed with mandatory access control. There are also parts taken from the Linux Security Module Framework[28] (LSM). The security mechanism can decide access control decisions based on configurations by the administrator in forms of labels. Labels are structured information regarding the security system that contain error messages and other security specific access control information. SELinux was at first developed as a set of patches to the existing Linux kernel but has become a more standalone distribution. In SELinux the policy decision logic and the policy enforcement logic have been completely separated from each other. The policy decision logic is placed in the security server, based on Flask, and the policy enforcement logic is implemented with the LSM framework interface. The Flask architecture has an access vector cache which provides help to minimize the computation time for the access control system. To make the system easier to use there are some modified user applications and applications to help the policy creation. A policy compiler, a file system labelling tool, role management tools and policy querying tools are some of the help tools to administrate the security system. User applications like *tar*, *login*, *sshd* and some file utilities have been modified for handling label information from the security system.[8] The figure 3.1 illustrates the structure of the SELinux framework and what specific kernel services it supports. The figure also illustrates the user processes and its interface to the system. The system interface is using different kernel services, which in their turn go through the LSM framework that is controlled by SELinux security system, which is based on different security policies. At this place the security decisions are made before letting the user processes perform the requested operations.

3.2 TrustedBSD and SEBSD

To add a security enhancement to an operating system is not an easy task. There are a lot of different concerns regarding the flexibility, maintenance, correctness and costs of implementing security enhancements to a system. This is why both Linux and the FreeBSD project started to work out a generic solution to these problems. Network Associates Laboratories and the TrustedBSD project started to implement a mandatory access control framework for the FreeBSD kernel. The TrustedBSD project addresses a lot of these issues, for example regarding flexibility for policy models and costs of using the security enhancement mechanisms in the system. The Security Enhanced BSD (SEBSD) project was started after the TrustedBSD project and is actually using the TrustedBSD framework combined with techniques from the Flask[22] implementation. The security server and access vector cache were taken from Flask to use for policy decisions. SEBSD project also added new structure for the kernel with new information for structures, this to have access control and at the same time generate valuable information. The figure 3.2 show

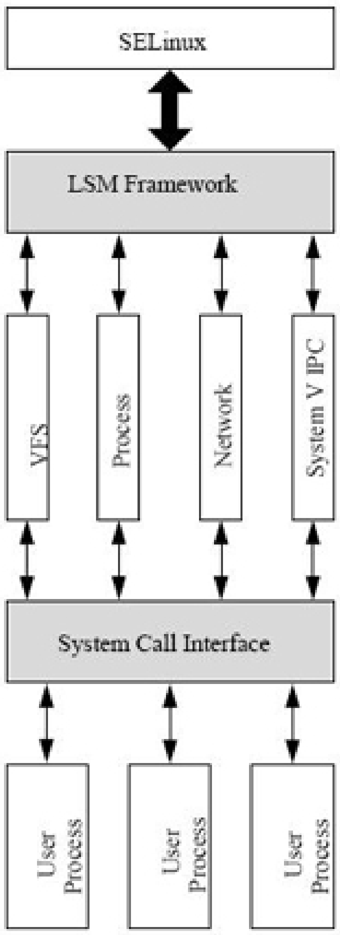


Figure 3.1: SELinux Framework structure. [8]

the structure of the SEBSD framework and what specific kernel services it supports. The figure shows the user processes and its interface to the system. The system interface is using different kernel services, which in their turn go through the MAC framework that is controlled by one or more security policies. The MAC framework is were the mandatory access control is managed. At this place the security decisions are made before letting the user processes perform the requested operations. There is no big difference between the SELinux and the TrustedBSD/SEBSD framework, which could be seen in the figures 3.1 and 3.2.

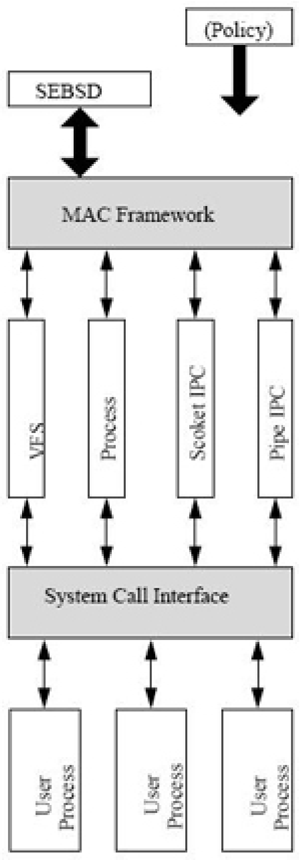


Figure 3.2: SEBSD Framework structure. [8]

Both TrustedBSD and SELinux have the same goals and problems to solve. But the main goal they have in common is to have modularity between the policy modules so the system does not get tied up to a specific set of policy models or implementations. TrustedBSD operates totally transparent for users and applications. Access control denials are reported as standard errors that the applications are generally taking care of in one way or another. Applications could be developed to take care of specific error reporting (labels) from the security system as well, but this is not necessary as general applications would not work in a TrustedBSD/SEBSD environment. There are some of the general system commands that have been ported with label aware implementations, for example *ls*, *ps* and *login*. These labels contain security system specific information regarding errors and other messages that the security system produce. TrustedBSD was developed to be tightly connected with the kernel to provide good support for systems with multiple CPUs. Multiple policy models may be loaded into the security server and the decisions are combined from all of the loaded policy models. The compositions of the decision making are performed by the TrustedBSD framework and not by the policy module developers. SEBSD is a ported version of SELinux and has about the same features. The only thing that was not ported was the labelling mapping system that SELinux uses, because of the different file system structure between Linux and UNIX, FreeBSD uses UNIX File System (UFS) or UNIX File System 2 (UFS2), while SELinux uses Extended file system (Ext) or Extended file system 2 (Ext2). Both SELinux and SEBSD are using Mandatory Access Control, Role Based Access Control, Type Enforcement and Multi-Level Security.[8]

3.3 RSBAC

RSBAC[34] (Rule Set Based Access Control) is an extension for the Linux kernels and it is based on the Generalized Framework for Access Control (GFAC)[24]. GFAC is a very general framework for access control that both RSBAC and SELinux use. RSBAC is open source and is reviewed by independent researchers. The extension provides a flexible system for access control and comes with several modules that implement different security policies.

RSBAC consist of three elements ADF (Access Decision Control Facility), AEF (Access Enforcement Facility) and ACI (Access Control Information), which is illustrated in figure 3.3. The elements act between the subject and the objects. The figure 3.3 shows how a process wants to access some system object, which could be a file, directory, device etc. The system call goes through the AEF (1) and this element checks the ACI for information regarding the access rights (2). AEF passes the request to ADF for decision (3), which refer to the ACI (4) and makes a decision that is passed back to the AEF (5). The decision is made and the process gets access or is denied access (6) depending on the decision that ADF made. If the access is enabled for the process, a notification message is sent from the AEF to ADF (7), which updates the ACI and sends back an acknowledgement to AEF that gives the process access to the requested object.

RSBAC is not only adding dynamic security policies for the system, but also adds a policy independent logging facility that has logging functionality for all used policy models. The events to be logged may be specified from the request type, user, executable and target file, FIFO, symlink, directory or device objects. The information that the log gathers are user, process, object and request information. The models can specify their own access control scheme for the logging administration that specifies how the logging should be performed.[34]

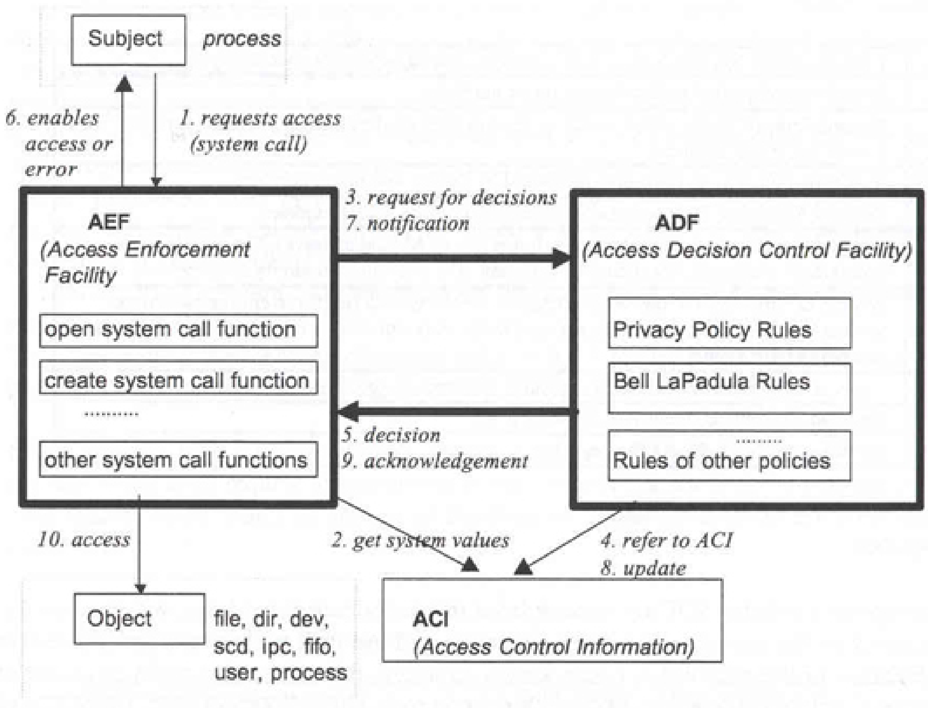

Subject
process
6. enables access or error
1. requests access (system call)
AEF
(Access Enforcement Facility
open system call function
create system call function
other system call functions
3. request for decisions
7. notification
ADF
(Access Decision Control Facility)
Privacy Policy Rules
Bell LaPadula Rules
Rules of other policies
5. decision
9. acknowledgement
10. access
2. get system values
4. refer to ACI
8. update
Object
file, dir, dev, scd, ipc, fifo, user, process
ACI
(Access Control Information)

Figure 3.3: RSBAC security mechanism structure. [34]

3.4 Summary

SELinux, which is based on Flask, is the Linux version of SEBSD. TrustedBSD is needed by SEBSD and is included in the newer versions of FreeBSD. The basics of these systems and security enhancements has been described in this chapter and forms a ground for the discussion about these systems in the project results in chapter 7.2. RSBAC has been explained in this chapter as it is an alternative to SELinux in Linux for security enhancements. The RSBAC system is discussed in the project result chapter as a possible alternative for SEBSD, if it becomes ported for BSD.

Chapter 4

Attacks and Defense

This chapter explains attacks to operating systems in general as well some defense tactics. The last part of the chapter explain vulnerability analysis tools and the tools used for the experiment in this project.

4.1 Attacks

Attacks fall in to four different categories, *who*, *goal*, *vulnerabilities* and *defense*. The categorie *who* are the ones that perform the attack and these attackers usually have a *goal*. The attackers use vulnerabilities in a system to get access and a *defense* system is needed to protect the system from these attacks. The people who do this can be subdivided into three other categories[21]:

- Misfeasors
- Masquerades
- clandestine users

Misfeasors are users that have authorized access to a system but misuse their rights on the system. This may be a user that tries to gain administrator privileges by attacking the operating system from their own account or change data they are not allowed to change.

Masquerades are people that do not have authorized access but have gained it by cracking a user's password and acting like the real users in order to perform an attack to the system from the inside. The *clandestine users* have gained their own access to a system by obtaining administrator access and than create their own access paths, e.g. creating new user accounts.

There are more ways to gain access to a system than these. People may interact with the hardware where the system is stationed. They may add new connections to the system or removing storage hardware to obtain information or administrator privileges.

Attackers of a system always have a goal to attack the system, it may not be with the intention to cause serious damage, but just conquer the system to prove what they can do. The *goals* an attacker may have can be the following[21]:

- Trophy grabbing
- Information theft
- Service theft
- Identity theft
- Tampering
- Denial of Service

An attacker that wants to prove his abilities is usually not doing any particular damages to the system but instead leaves a trophy showing that he managed to attack the system. A common *trophy* is to change the Webpage if the system is running an http server. *Information theft* is another goal for an attacker. These attackers are looking for passwords, credit cards numbers and other sensitive information. Attackers can attack a system just to gain access to its resources. This is called *service theft* and is usually done to provide computers to do new attacks from, or used for storage of illegal software. There are also attackers that want to obtain other people's resources such as e-mails and bank accounts. This is *Identity theft* and is usually used to get money or other privileges that the attacked person has. *Tampering* is when people are not stealing information but instead changing the information. An employee that changes his salary in the database without anyone noticing it is just one example of tampering.

Denial of Service (DoS) attacks are rather common nowadays. This is one of the most dangerous attacks to a system. It may collapse the system and data may be lost that was meant to be sent to the system. The meaning of this attack type is flooding high amount of data to a system, which the system cannot handle. When there are several computers that perform these attacks with the same target it is called Distributed Denial of Service attack (DDoS). These attacks are usually performed by systems that have been high-jacked by some intruder.

Some common attacks to an operating system are[21]:

- **Backdoors** - Create an unauthorized access point to an system
- **IP Spoofing** - Forge the IP address
- **Masquerading** - Posing as an authorized user
- **Packet sniffing** - Read data packets sent to and from the system
- **Replay attacks** - Record data packets and re-send them to gain access
- **Security audit tools** - Scan a system for vulnerabilities to exploit
- **SYN Flooding** - Send TCP initial synchronization without acknowledge them, causing a lot of open sockets on the attacked system.
- **Trojan horses** - Software that seems useful but has in addition a hidden malicious function.

- **Worms** - A program that replicates itself in a network.
- **Virus** - A program that can infect other programs by modifying them to include an, possibly evolved, copy of itself.
- **Buffer overflows** - A program or attacker writes data beyond the allocated buffer in memory and may execute other malicious code from memory.

These attacks are usually combined and most of them require network connection to the system. The attacks above are already known attacks, which are easier to protect the system from. But attackers may find new ways to attack a system.

The causes of the category *vulnerability* may be subdivided into five new categories[21]:

- Implicit trust
- Configuration error
- information leakage
- Weak design
- Carelessness

The first vulnerability, *implicit trust*, is if a person is accepted without any question of identity, for example that the system does not require any further authentication if the IP address is what was expected. This may be forged with social engineering, IP spoofing, masquerading, Trojan etc. *Configuration errors* may lead to security holes in a system, which attackers can use to exploit the system. An attacker can give out information (*information leakage*) to others to show the weaknesses of a system and open for attacks. Also a common mistake in software engineering is to have a *weak design* when it comes to security. The software may function correct, but can have buffers that may be exploited with buffer overflow. The category *carelessness* includes systems that have been badly patched. Those are systems that run old software with known security breaches. A system where users have chosen weak passwords or perhaps even the system administrator has chosen a weak password is another example or carelessness.

4.2 Defense Tactics

The previous chapter explained different attacks to a system, but a system needs to have some *defense* against the attacks. The defense category can be divided into five strategies[21]:

- Obfuscation
- Authentication and authorization
- Monitoring and auditing
- Up to date software
- Education and enforcement

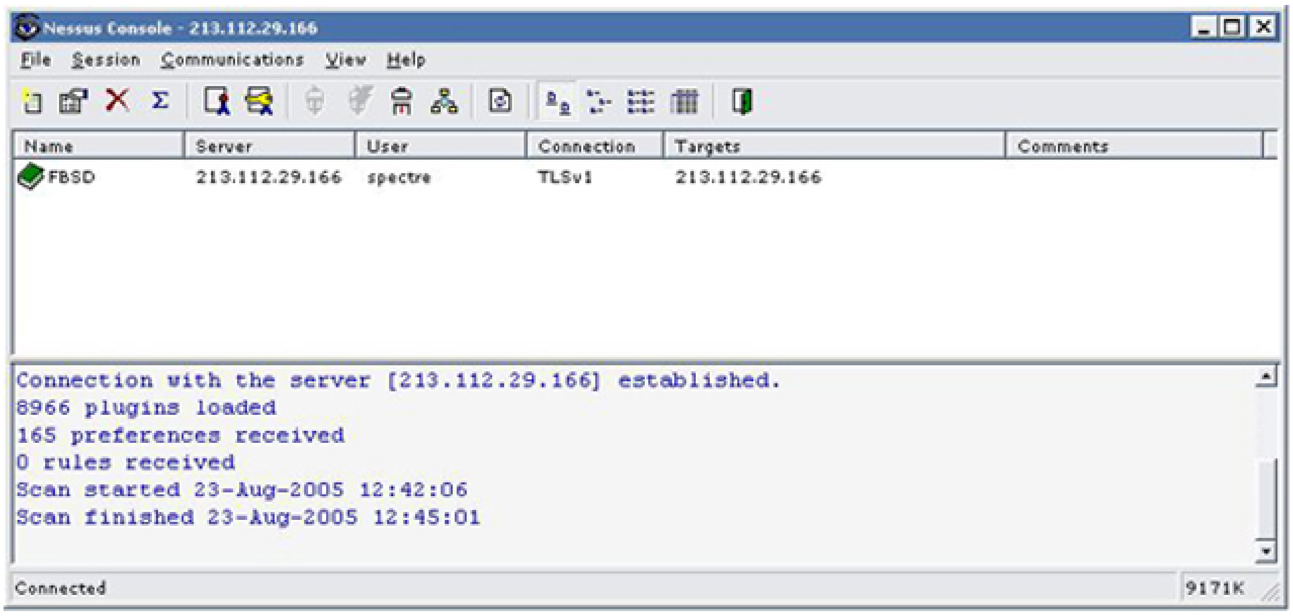
These five categories are what an administrator can use to defend a system against attackers. Obfuscation is hiding of information, usually applied with cryptography. The first three categories have been discussed in the previous chapter (chapter 2). To avoid carelessness in the system, *up to date software* is very important. It is important to always have the latest updates of software that runs on the system, upgrading the system if it is necessary and to review the processes that the system executes. New versions of software has bugs corrected, which before the update could have been exploitable. The software that the systems are using should be developed with security in mind. So *education and enforcement* is important in that way so new software is developed from scratch with high security. [21] Other tools that could be used are vulnerability testing tools that may prevent attacks, such tools are described in the next section 4.3.

4.3 Vulnerbility Analysis Tools

There are a lot of different vulnerability analysis tools that could scan a system for possible flaws and exploitable bugs. The problem with these tools is that they need to be constantly updated to find all security breaches. Nessus[42] and SARA[9] are two of the most updated and used applications for security evalutation of UNIX systems, which were for this reason chosen as vulnerability testing tools in this project. Beside these two tools, there are a lot of other vulnerability testing tools that have been combined with intrusion detection systems. Examples of such tools are *Sentarus*[25] and *ISS*[44]. Sentarus has not been updated for a very long time which makes it pretty useless for this sort of test and ISS is not freeware, which means that it needs to be bought and registered. If this was not the case ISS would have been used for another vulnerability test on the selected systems.

4.3.1 Nessus

Nessus is an open-source vulnerability scanner and is used by many organizations. The project was started in 1998 by Renaud Deraison and was aiming to be a free and powerful remote security scanner. Nessus provides both remote and local security checks. It checks for flaws in both the system as well as in specific applications and services that are running on the system. Nessus provides network scanning as well as single host scanning. The developers focus on developing up-to-date security checks for the recent security holes. Nessus consists of portscanners, tests of pseudonumber generators and other parts which could be used stand-alone for testing. Nessus makes it easy with its collection of security testing applications in form of scripts. Nessus is installed as a daemon that runs in the background on a targeted system. A client program is used to connect to this daemon and execute the actual vulnerability test. It is the client that decides what plug-ins to use and how to scan the target system. These plug-ins are vulnerability testing scripts that execute on the server. These scripts may have been written by contributors to Nessus or the developers of Nessus.[42] The picture 4.1 show the NessusWX client user interface that executes under Windows[6].


Nessus Console - 213.112.29.166
File Session Communications View Help
Connected
9171K

Name	Server	User	Connection	Targets	Comments
FBSD	213.112.29.166	spectre	TLSv1	213.112.29.166	

```
Connection with the server [213.112.29.166] established.
8966 plugins loaded
165 preferences received
0 rules received
Scan started 23-Aug-2005 12:42:06
Scan finished 23-Aug-2005 12:45:01
```

Figure 4.1: The Nessus client user interface after a scan has been performed.

4.3.2 SARA

SARA stands for Security Auditor's Research Assistant and is a tool for auditing the system security. SARA is based on the other system security tool named SATAN. SARA operates under UNIX, Linux, MAC OS/X[20] and Microsoft Windows. It is free and with open source code. SARA may be used not as only a complete system analysis tool but as well for specific applications. It supports security analysis for SAMBA, NMAP and other software. It is pretty much like Nessus even though it seems a little more out of date with security issues. The developers of SARA state that they try to update SARA twice every month, but this may not be enough for the ever-growing system exploits.[9]

4.4 Summary

Common attacks and how to defend a system against them was the main goal of this chapter. Some common attacks and the general purpose of the attacks were described. The second part of the chapter describes some basic defense tactics that could be used in an operating system. The vulnerability analysis tools that were used in the vulnerability experiment has been shortly described and a general introduction to vulnerability analysis tools was given. It is good to know the target of protection. Defending a target is not just something that has to be done but has to be done for special reasons. If the system stores a lot of secret or important information there could be many attacks from intruders that want to get hold of this information, therefore heavy security should be applied. A web-server could be a target for trophy grabbing as it has web pages open for the public. This are just two examples of attacks to different systems in different environments. The chapter has explained different attacks and an introduction how to protect a system against them.

Chapter 5

Comparison of Security

This chapter provides an introduction to the tested operating systems OpenBSD, FreeBSD and TrustedBSD. Each security area will then be discussed with the selected systems as targets and a short review of the comparison follows.

5.1 Introduction

5.1.1 OpenBSD

The founder of OpenBSD was Theo de Raadt, who was a developer for the NetBSD project back in 1990s. OpenBSD diverged from NetBSD in November 1995. As the OpenBSD project was developed in Canada the project was not under the influence of United States export laws, which made it legal to use strong cryptography in OpenBSD.

OpenBSD is a distribution based on BSD 4.4. The developers state that OpenBSD is the most secure UNIX-like operating system[2], whether this is true will be discussed in the following chapters. The goals of the OpenBSD project are to have correctness, security, standardization and portability. The system is only dealing with encrypted data over network and is said to be no non-secure connections in a default installation of the system. The cryptography is exported with OpenBSD as default. The development team has a group that audits all files for the system and analyze the code for not just possible exploits but for code flaws that could be exploited in the future. As the files are audited by several professional developers the source code is stated to be very correct and secure. With this technique the exploits that may be found in other systems are usually already fixed in OpenBSD by the time they are found in the other systems. OpenBSD is said to be *secure by default*, which means that the novice users do not have to learn everything directly. It is also said that all services that are not needed to run the system have been disabled as default, also this will be investigated. As the administrator is learning the system, the administrator is also learning about security considerations in the system, according to the developers.[2]

5.1.2 FreeBSD

The FreeBSD project started out as a patch for the 386BSD project. The patch was in the end large and acted as an own operating system, which was later renamed to *FreeBSD*.

FreeBSD is just like OpenBSD a UNIX distribution based on BSD 4.4. The distribution is supporting a wide range of different hardware platforms. The goal of the FreeBSD project is to provide an operating system that may be used for any purpose without any obligations for the user. FreeBSD is developed under the GNU General Public License (GPL) and some parts are developed under their own FreeBSD License. This license is used so other people cannot claim that it was their project or that FreeBSD project has stolen their code. The source code for FreeBSD is open for public in the terms of GPL and FreeBSD License. FreeBSD is designed to provide a full-featured and stable environment for applications. It is said to be suitable for both a workstation and a server.[1]

5.1.3 TrustedBSD

TrustedBSD was first a standalone distribution that was built from the FreeBSD distribution. The project was creating a framework that supported access control lists and security policies based on mandatory access control, for example the MLS confidentiality, Type Enforcement and the SEBSD module supporting other security policies from SELinux. In later versions of FreeBSD the framework has been integrated into the FreeBSD distribution. The framework is adding trusted operating system extensions to FreeBSD. The main features of the TrustedBSD are to add extensible and audited authorization framework to support access control modules and also to have a centralized policy management. It allows support for a variety of access control methods, which adds a special form of labelling for domains and types as well a special access control decision method. The SELinux (see chapter 3.2) security models have been ported to FreeBSD, called SEBSD, and the security models can be enforced with the SEBSD module trough the TrustedBSD framework. TrustedBSD improves system privilege to reduce the risk of common system management. The framework also adds support for better event auditing and single-host intrusion detection system to monitor security events.[3] Chapter 2.1 explains more in detailed what the difference between a regular operating system and a trusted operating system is.

5.2 Authentication

The authentication in all tested systems is based on login name and password, both through console and remote connection services like SSH. Having authentication to the system with login and password makes the system a lot more vulnerable for social attacks where other people may get hold of the password and login. Often users write down their passwords to remember them. A better solution would be to use one time passwords as default, at least for the super user account (root). Both FreeBSD and OpenBSD provide one time password support for users, which means that the users get a key output at the login session and the users generates the passwords from that key in another application. There is also support for rlogin and telnet but these services do not provide any encryption of the connection and cannot be trusted, they are not started as default in any of the systems.

What should be noticed is that OpenBSD has configured SSH by default to let *root* login over network with SSH. FreeBSD does not let *root* login over an SSH connection by default, which is a lot more secure. This is more secure because it is not hard to see if a new connection over SSH is initialized by scanning the network with a packet sniffer. If then root login at the beginning of the SSH session, the attacker may try to decrypt the first packages as he know that the root password is hidden there. When login with *root* is performed through SSH on the OpenBSD system there is a warning displayed in the console that the super user, *root*, should not login through network connections but instead use the *su* (Super-User) command. This is done by logging in with another user account that is in the same group as *root* and than use the *su* command to change to super user privileges. By the time the system administrator sees this warning message it could already be too late and the system would be open for attacks. The root login should be disabled for SSH in OpenBSD by default. As mentioned earlier the developers of OpenBSD state that they aim for very high security regarding remote connections, but letting *root* login through SSH is not following that statement.

Kerberos is supported in all of the tested systems and in FreeBSD version 5.1 or later and OpenBSD 3.6, Kerberos version V is only included. Kerberos V is not secure for binary streams over telnet or rlogin, which should not be used. A combination of Kerberos with SSH is the most secure way to connect remotely. All of the tested systems could use pluggable authentication module (PAM)[27] to get support for smart cards, biometric devices and other authentication techniques but there is no default support for these devices in neither of the systems. Authentication techniques, other than login combined with a password, are getting much more common nowadays. Support for at least smart cards should be included as default in all of the systems to get a wider usage of the systems in different environments. The authentication methods in the systems should also be easier to install instead of letting the system administrator edit a lot of configuration files that may be hard to configure correctly.

5.3 Access Control

Discretionary Access Control (DAC) is the standard to enforce the permissions on FreeBSD. This is not a very secure access control scheme as it is vulnerable for Trojans and other malicious software. For example a Trojan could change the privileges of important files to reject the original owner of the files to read them. The Trojan could then set the privileges of the files to allow other users to read them, who should not be able to read them. A system using DAC is much more sensitive for attacks than a system that uses MAC or MAC combined with DAC. The various modules, based on MAC, which the TrustedBSD framework provides in FreeBSD, could be used to protect the network and file systems to block users from using different types of services or even block specific socket ports on the system. The best security is achieved by using several of these modules together. But there are downsides from the administrator's point of view because the administrator needs to set the network access control user by user and all other rules manually. This creates a lot of extra work to get a secure system. This might be easier to perform by creating scripts that do this automatically. But overall it is a lot more work for the administrator.

A wrong configuration of the framework could also stop authorized users to get in to the system, and perhaps even lock out the admin from the system. But if the framework is correct configured this extra work will hopefully pay off in a much more secure system There is several known security models used in the framework, LOMAC, Multi Level Security (MLS), fixed and floating Biba[37] integrity model. The LOMAC module is almost like Biba but permits access to lower integrity objects. These are the standard modules that are included with the MAC framework. For a small scaled system without too many users these models seem more than enough but for a system with a lot more users the SEBSD module could be installed, which adds more security models based on SELinux/Flask implementation. SEBSD adds Role Based Access Control (RBAC) for example. RBAC is much more appropriate for commercial applications then Biba and MLS, which usually suits for military applications. The TrustedBSD framework is not used as default in FreeBSD and should be activated as an option to the kernel before it could be used. During the installation of the system there should be an option if the installation should include the framework and if so, compile the kernel automatically from the installation options given by the system administrator.

The Access Control Lists included in the newer versions of FreeBSD extend the standard UNIX permission model as chapter 2.3 "Authentication" described. The ACL is included in the generic compilation of the kernel. Extended Attributes are needed for ACL to function. These attributes are extra information for files and directories in the file system. For example if a file has ACL combined with it, there is + sign after the access control rights in the file information. Extended attributes for ACL are supported for the new version of the UNIX file system, UFS2. A lot more configurations are needed to use extended ACL attributes with a UFS1 file system, so UFS2 is the recommended file system to use with extended ACL attributes. When ACL is activated for a file system in FreeBSD there cannot be any change to the disposition of the file system when it is in use. There is a flag, called super block flag that could be set for a file system. Once this flag is set the file system will always boot with ACL even if the file system is removed from *fstab* (configuration file for device mount). This is to prevent that the file system is accidentally booted without ACL enabled. In OpenBSD there is currently no Access Control List support for the file system. The developers seem to have no plans for this to come. The OpenBSD system is using the three fields protection scheme that were explained in detailed in chapter 2.2 "Access Control". This makes the OpenBSD much more vulnerable for attacks from inside the system by users or masquerading users. It would be enough to have a Trojan on a user account to change privileges for the user's files. As OpenBSD also lacks mandatory access control it is not suitable for many users that should have different privileges to files, devices and directories. [1][2]

5.4 Cryptography

FreeBSD is using a library (libcrypt.a) that supports configurable password authentication hashing. The library supports the encryption and hashing algorithms DES, MD5 and Blowfish. The default algorithm for encryption of passwords is MD5. US law restricts the source code of the DES algorithm to be exported which the FreeBSD developers has solved by letting users in US use DES encryption and users outside US use MD5. This is also why MD5 is used as default encryption algorithm in FreeBSD. MD5 is said to be more secure [1] than DES but for some compatibility reasons DES is offered as a choice. The algorithms are used in both kernels as well as user land applications. There is difference between MD5 and DES even if they are used for the same purpose in this case. DES is an encryption algorithm and MD5 is a one-way hash algorithm. MD5 is creating a hash code for the password and stores the hashed value into password lists. The system itself only knows the hash code and not the plaintext password.

Recently there was a collision detected for MD5, which means that MD5 was considered broken. A collision appears when same MD5 value is returned from different input. The way this collision detection was performed does not reflect on the security of using MD5 as a password encryption technique in UNIX systems. But even if this is the case for this specific collision detection another encryption technique should be used for the passwords in the system. Next time a flaw in MD5 is found, the system may be vulnerable for attacks. Blowfish is a good alternative for MD5 and has not yet been broken. Some parts of the algorithm have been broken but not the whole algorithm and it is considered as a trusted encryption algorithm at this time. DES is a symmetric encryption algorithm and is only using 56 bit keys and has recently been broken within 22 hours, so 56 bit keys is vulnerable to exhaustive brute-force search of the key space. Another symmetric encryption algorithm AES (Advanced Encryption Standard) could be an alternative for DES as AES is using up to 256 bit keys in contrast to 56 bit keys and is considered as secure.

FreeBSD and OpenBSD are using OpenSSH, which is supporting SSH version 1, 1.5 and

2. All cryptographic restrictive components in the client have been removed and support for Kerberos authentication and ticket passing is included. To have a secure SSH session, SSH protocol version 2 should be used. OpenSSH have support for Blowfish, 3DES[41], Arcfour and AES.

As the OpenBSD project is based in Canada there are no laws against exportation of cryptography in software. OpenBSD does not use any patented cryptography only free cryptography is used. The OpenBSD project is using IPsec and Kerberos V for different purposes in the system. The cryptography is used in several areas;

- ssh (OpenSSH)
- PseudoRandom Number Generators (PRNG)
- Cryptographic Hash Functions
- Cryptographic Transforms
- Cryptographic Hardware Support.

As mentioned earlier in this chapter OpenSSH is used in both FreeBSD and OpenBSD as the default SSH package. The PRNG is used to provide applications with randomly generated numbers that is used for different security relevant purposes in the system. The PRNG is constructed in a way that the same input gives the same output. An attacker should not be able to guess the next sequence of numbers by looking at previous output. Some applications for PRNG are random padding in IPsec, RPC transaction IDs, PIDs of processes etc. In OpenBSD the algorithms *MD5*, *SHA-1* and *RIPEMD-160*[41] are used for this purpose. The Cryptographic Transform is used to encrypt and decrypt data. In OpenBSD the algorithms for this purpose are *DES*, *3DES*, *Blowfish* and *Cast*. These are used in both kernels as well as user programs. The last area of cryptography in OpenBSD is the Hardware Support with cryptography where there is different hardware that needs support for cryptography. MD5 is used in OpenBSD as well and the discussion earlier in this chapter explained why MD5 should not be used as a default hash code generator (encryption) technique. AES with 128 bit key length is used in some areas like hardware encryption and the secure socket layer (SSL)[45]. SHA-1 and RIPEMD-160 are two hash code algorithms like MD5. But SHA-1 has recently been broken and should not be used for password encryption. RIPEMD-160 has not yet been broken and could be a fair alternative for both MD5 and SHA-1. But the same weakness that has been found in MD5 and SHA-1 has also been discovered in RIPEMD-160 but it has still not been broken.

IPSec is included in both FreeBSD and OpenBSD but needs to be compiled into the kernel before it could be used. IPSec should be an alternative in the installation process that let the administrator choose if this should be installed by default. A novice system administrator may forget IPSec if it is not an option during the installation.

5.5 Auditing and Logging

Both FreeBSD and OpenBSD are using the *syslog* service to provide log functionality to the system. As described in the chapter "Auditing and Logging" (Chapter 2.4) *syslog* provides a logging service for all applications that are used in the system and logs can be created on local machine as well as over network on another system. All logs are plaintext files, that makes it easy to read the logs and to use scripts or programs like awk and sed to process the log files and give a specific output. Both systems have the same logging provided. Depending on what system configuration is in use and what applications are used on the system, the logs may look different. The logs are created in the file system directory */var/log/*. In the default installation of the system there is already a default configuration for the *syslog* service in the file *syslog.conf* which is located in */etc/* on both systems. The system administrator can in the *syslog* configuration file tell *syslog* what to log with regard to what files. Examples of a *syslog* configuration file (syslog.conf):

- *.notice;kern.debug;lpr.info;mail.crit;news.err /var/log/messages

- auth.* /var/log/auth.log
- mail.info /var/log/maillog

In this example the log messages that the kernel, printer, mail and news services produce are saved into */var/log/messages* and all authentication to the system through terminal as well as remote connections are logged in the file */var/log/auth.log*. Information that the mail service provides is logged into another file called *maillog* in the same file system directory as the other log files. In both systems this log service is used and could be configured by the system administrator to log what the administrator wants and thinks is important. This is a very flexible way of handle the logging of the system and its services. These log messages could also be specified to output to console, to all users or to web pages FreeBSD has nine layers of importance for the *syslog* service compared to the default of eight layers:

- **emerg** - Most critical messages, system is very unstable.
- **alert** - Almost as bad as emerg level, but system could be still operating.
- **crit** - hardware errors or serious software issues. System could be operating.
- **err** -Errors that don't crash the system but should be fixed.
- **warning** - Different warnings concern applications and system.
- **notice** - Information that should be logged if needed to analyse behaviour of the system.
- **info** - System information about events in the system.
- **debug** - Information for developers about what a program do during execution.
- **none** - Does not log anything for the specified service.

These nine layers specify how to output to logs, terminals or administrator. While *emerg* layer output the messages to all terminals and also to log file, the *info* just save to log for further usage if needed. The critical layers are part of the auditing where the administrator monitors if any high security breach occurs. *Syslog* only logs what the applications are implemented to log, so the auditing is as good as the applications are in logging events. OpenBSD is not implementing any further approach for the *syslog* service and using the default scheme. The ninth layer *none* that FreeBSD adds to the *syslog* service is used for wildcard entries that do not need any logging. There is much logging for most services and applications in both systems but there is not any good tool to process the logs for example attacks. Some logging is displayed to the standard output (usually the console) but if login attempts fail there is only logging to file, except for *root* login failure, which is displayed to standard output. A default log surfing tool would be very suitable for both systems, where logs could be overviewed in real-time.

If the SEBSD module is loaded in FreeBSD there is a lot more logging performed to standard output. Every event that fails is displayed. This is a nice feature as it is much easier to configure the policy modules if there are messages displayed for events that fail. These messages could be redirected to a log file when configuration of the modules is complete.

5.6 Firewalls

FreeBSD has as default three different types of firewalls included in the system. *IPFILTER* (IPF) , *IPFIREWALL* (IPFW) and *Packet Filter* (PF) are the firewall packages that are included. Why there are three different built-in firewall packages in FreeBSD is because there may be different demands and requirements for a system depending on which environment it is going to operate in.

Packet Filter is OpenBSD's firewall system, which has been ported to FreeBSD. Packet filter is a fully featured firewall with Quality of Service functionality. The IPF firewall contains the basics for creating a firewall framework but also supports modules for creating a really strong firewall. IPF is also having a built in ftp proxy and is considered to have less complicated rule set. The last firewall IPFW is the FreeBSD projects own firewall created by the project members. IPFW is aiming for the professional users because of its highly sophisticated rule sets and packet filtering techniques. Except these three firewall packages, FreeBSD has two default bandwidth controllers that control the bandwidth usage in the system, *altq* (Alternate Queuing) and *dummynet*. *Altq* provides Quality of Service (QoS) to different applications, which means that the selected applications get a guaranteed bandwidth. This is done by rule sets. These bandwidth controllers are used together with the firewall packages to create an environment where to reject a single user from using full bandwidth. Without this controller a user could make other users bandwith low. The three firewalls are all using rule sets to control the access of packets in and out in the system from a network. The differences between them are that they do it in a different way and have a different rule set syntax as well as being more or less easy to configure for the system administrator. The different ways they choose which packet filtering rule to use may be for example first matching rule, where the first rule that match the incoming packet is used to filter the packet, or last matching rule, where the last matching rule is used to filter the packet. And these rules have different syntax depending on which firewall that is used.

The firewalls are included but need to be enabled by the system administrator before they could be used. The bandwidth controllers need to be compiled into the kernel before they could be used in the system. OpenBSD is using, as mentioned earlier, *Packet Filter* (PF) to filter TCP/IP traffic and to do network address translation (NAT). PF also has integrated quality of service and can therefore control the bandwidth and packet prioritisation. PF is included as default in the OpenBSD release, since version 3.0. To get the PF firewall to work, it needs to be enabled in the boot configuration file (rc.conf.local). This is also the file that should be edited on the FreeBSD system to get the firewall to work. The rule sets that the firewalls, on both systems uses, should be configured by the system administrator as they have no default configuration that works for all systems.[2][1]

What should be noticed is that during the FreeBSD installation there is a question asking how high security there should be for the system, but it never mentions if there is a firewall installed by default. It should be easier to install a firewall during the installation for both systems. After having a system up for about one day there had already been login attacks to the system, which could have been blocked with a correct configuration of a firewall. The freedom of choice for firewalls in FreeBSD is good as the system administrators may have different skill levels and the different firewalls features different types of configurations. A system may also need a heavy firewall setup and then the IPFW could be used. OpenBSD only features their own developed firewall, but as its including the important parts it is enough with just PF, even if it could be harder to configure for the system administrator.

5.7 Discussion

All tested systems have login name combined with a password as default authentication method. An alternative for this would be to use one-time passwords that at least make the accounts more secure against social engineering attacks. The only system for remote login should be SSH and even better combined with Kerberos. SSH is dealing with encrypted data transfers, which makes the connections a lot more secure against sniffing and Kerberos has ticket authentication, which makes the authentication more secure and makes spoofing harder to perform for attackers. There is no default support for other authentication systems like smart-cards and biometrics (e.g fingerprints) logins in neither of the systems. Smart-cards should at least be supported by default as it is rather common nowadays.

As the FreeBSD project adds the TrustedBSD framework to their distribution, it is now a choice to use access control lists and policy based access control rules. OpenBSD does not support access control lists or mandatory access control. The drawback with the framework is that it is hard to configure and could make the system less secure if the configuration is erroneous. The default installation of TrustedBSD framework has enough policy modules for a simple setup of MAC in the system. Having the SEBSD addon installed adds even more security models to the framework, which opens up a more wide use of the system with better support for MLS and DTE. As the framework is a complex system, the OpenBSD developers do not seem to adapt it in the nearest future, which makes the OpenBSD distribution less useful in multi-user environments.

The cryptography used in the systems is basically the same, but OpenBSD is not only using encrypted data for remote connections in the system but encryption for hardware as well. FreeBSD is using MD5 as default encryption of passwords in the system while OpenBSD uses MD5, SHA-1 and RIPEMD-160 for this purpose. As OpenBSD is not restricted by laws for the use of cryptography in their system they could use more secure algorithms than FreeBSD, which has restrictions by the US law. Both systems have DES and Blowfish as choice for encryption, but this is usually just for compatibility reasons with old systems. OpenBSD are using 3DES as alternative. OpenSSH, which is developed by the OpenBSD developer team, is used in both systems and has support for better encryption methods than the systems, which seems rather strange. MD5, DES, SHA-1 and RIPEMD-160 have all been either broken or have shown weaknesses. Blowfish and AES seem like a better alternative for the encryption method of, for example passwords.

The auditing and logging in both of the systems are based on the syslog service. Both systems are logging the same information by default, like system login information. The difference between the systems is that FreeBSD adds extensions to syslog with one extra layer. There is still not a big difference between the logging information. Both systems lack a good tool for log viewing by default. Such tools could be installed afterwards but could be good to have installed with the base installation.

There are three firewalls with QoS extensions available in FreeBSD compared to OpenBSD, which only has one firewall as default. Which firewall of the three included in FreeBSD to use, is up to the system administrator and his requirements. Both systems are including Packet Filter (PF) firewall, which also adds QoS to the system. FreeBSD adds more freedom for the system administrator what firewall to choose compared to OpenBSD that only includes PF. This could affect the system security in the means of system administrator skill levels. If the system administrator has low skills and configuring the PF firewall incorrect there could be bad consequences to the system and its users. But having a choice for other firewalls where the syntax for configuration is easier there could lead to much better security.

5.8 Summary

This chapter has compared the selected operating systems in the different areas regarding computer security. The basic differences has been covered and a discussion concerning the results has been given in the end of this chapter. The comparison in this chapter is more theoretical than practical, even if some parts are based on tests with knowledge about the systems. For example the root login through SSH was tested on both systems. This chapter could be seen as the first part of the comparison where the other part is the practical vulnerability experiment in chapter 6.

The table 5.1 illustrates a rather abstract summary over the different areas for each selected system. The security is measured in three different levels, low, medium and high.

Abstract System Security Overview					
	Authentication	Access Control	Cryptography	Auditing & Logging	Firewalls
TrustedBSD	Medium: password	High: MAC/TE/RBAC	Low: MD5/DES	High: syslog/AC-logging	High: IPFW/IPF/PF
FreeBSD	Medium: password	Medium: DAC/ACL	Low: MD5/DES	Medium: syslog	High: IPFW/IPF/PF
OpenBSD	Medium: password	Low: DAC	High: AES	Medium: syslog	High: PF

Table 5.1: Summary of the security in the systems.

Chapter 6

Vulnerability Experiment

This chapter explains how the experiment was performed and what configuration was used. The results are documented and briefly discussed.

6.1 Experiment Configuration

The versions of FreeBSD that were tested were 5.3-BETA2 and 5.4 both with a generic kernel. Since a new release of FreeBSD was published during the project the two versions were tested. Testing a new version is a good way to see if the security holes in the old version were handled. TrustedBSD/SEBSD was not tested because it would give the same results as testing a regular FreeBSD system, this is explained in more detail in chapter 7.1. The 5.3-BETA2 installation of the FreeBSD system had *Emacs* installed as the default text editor. The new version, version 5.4, was using the *edit* text editor as default and no extra software installed. The reason for having *Emacs* installed on the first system were to have faster and easier configuration of files. The other system that was tested was OpenBSD 3.6 without any extra software installed. Some configurations were made to both systems to make everything work that was necessary for the test. The network was configured on both systems as well as basic language and keyboard set-ups.

The hardware configuration has no effect on this particular Nessus vulnerability test but may be necessary if the test should be recreated. The computer that was used for the test has the following hardware configuration:

- **CPU :** 400MHz Intel Pentium II MMX
- **RAM :** 128MB SDRAM
- **HDD :** 8GB Wester Digital, Caviar 38400
- **HDD :** 5GB Quantum Fireball
- **HDD :** 1.2GB Seagate
- **GFC :** Geforce2 7700 64MB PRO
- **MOB :** Intel 440BX/M
- **NIC :** 10/100 RealTek Ethernet
- **KBD :** Compaq KB-9965

To connect to the Nessus server on the target system the client NessusWX (version 1.4.5) was used. This client was installed on a Microsoft Windows XP system because of the user-friendly interface of this particular client. It does not really matter what client to use. The client for the Nessus server was configured to scan for TCP ports starting at 0 up to 1024, which are the most common used ports and are restricted to specific applications. Other configurations for the scan were to scan for SYN vulnerability. All other plug-ins that Nessus provide were used during the test. The Nessus server was of version 2.2.5 for UNIX systems. It was configured to let users login with password and not by certificate. There were no other configurations made to the Nessus server. The other vulnerability scanner that was used was SARA and the current version at this time was 6.0.6. No actual configurations to SARA were made. SARA was used on FreeBSD 5.4 and OpenBSD 3.6.

6.2 FreeBSD

6.2.1 Open Ports

The tables 6.1 and 6.2 show the ports in FreeBSD 5.3-BETA2 and FreeBSD 5.4 that were open when the system was portscanned with Nessus. The tables show the open port, the protocol type for the port, service and a short description of the service. The ports marked with *unknown* are services that the portscanner did not recognize. Security issues of the ports and services is explained in chapter 6.2.5.

FreeBSD v5.3-BETA2			
Port Number	Type	Service	Explanation
22	TCP	sshd	Secure SHell
111	TCP	sunrpc	SUN Remote Procedure Call
948	TCP	unknown	unknown
949	TCP	unknown	unknown
2049	TCP	nfsd	Network File System

Table 6.1: Open ports for the FreeBSD 5.3-BETA2 distribution.

FreeBSD v5.4			
Port Number	Type	Service	Explanation
22	TCP	sshd	Secure SHell
111	TCP	sunrpc	SUN Remote Procedure Call

Table 6.2: Open ports for the FreeBSD 5.4 distribution.

6.2.2 Security Risks (FreeBSD 5.3-BETA2)

Name: tiff Library, version 3.6.1.2

Risk: High

Nessus Output: "The remote version of this software is vulnerable to numerous integer overflow conditions which may allow a remote attacker to execute arbitrary code on the

remote host."
Explanation: tiff is a set of tools and libraries to handle .TIFF files, which is a image format.

Name: Fetch utility
Risk: High
Nessus Output: "There is an integer overflow condition in the processing of HTTP headers which may result in a buffer overflow. An attacker may exploit this flaw to execute arbitrary commands on the remote host. To exploit this flaw, an attacker would need to lure a victim on the remote host into downloading a URL from a malicious web server using this utility."
Explanation: Fetch is a command-line tool used in FreeBSD port collection that is used for retrieving data at a given URL.

Name: cups-base, version 1.1.21
Risk: Medium
Nessus Output: "There is a flaw in the remote version of this package which may allow an attacker to crash the queue browser service by sending a zero-length UDP message to the remote host."
Explanation: CUPS is the *Common UNIX Printing System*, a replacement for lpr. Attackers can fabricate UDP packages that contain no data and send them to this service on the remote system. By doing this they may crash the service and gain root access to the server.

6.2.3 Security Risks (FreeBSD 5.4)

There were no high security risks found in this version of FreeBSD. This may be a false positive result as it could be the Nessus scanner that has old scripts and missing some new vulnerabilities information, if there is any. It could also be true as it is a new version of the system that has not been used for as long time as the previous version of the system.

6.2.4 SARA Vulnerability Test

The services that were running according to SARA were *bootpc*, *smtp*, *SSH*, *syslog.* The output for SARA was not as detailed as the Nessus output, which shows the ports for the different services as well as which protocol the services use.

Name: smtp
SARA Output: "smtp relay."
Explanation: SARA found possible exploitable relay service in the SMTP server. If the relay service is not secured, spammers could easily send spam mails through the SMTP server from any host. If this is the case the server may be blacklisted on many other SMTP servers and these servers would reject the local server's e-mails.

6.2.5 Summary and Discussion

The first open port was the *sshd* service, which is the most common remote access protocol to use. There was no vulnerability found on this service. The *nfsd* (network file system daemon) is used but is not necessary on the system. *nfsd* is one of the most attacked services because it is widely used to share files over network with. *sunrpc* service is needed by the system for remote procedure calls in the system, no flaws detected for this version. There were two open ports, with no named application using them. The system does not have X-Windows system installed that is needed to view graphical files like images. Why the tiff library is installed may be to support a graphical version of the text editor *Emacs* that was installed as default for text editor. Both fetch and tiff library have integer overflow flaws that are open for exploits. Usually this is patched by applying a new version of the libraries to the system. Same idea with the other vulnerabilities, upgrading for new versions will usually solve the problem, as these are already known issues.

6.3 OpenBSD

6.3.1 Open Ports

The table 6.3 show the ports in OpenBSD 3.6 that were open when the system was portscanned with Nessus. The table show the port number that was open, what type of protocol it uses, type of service and a short description of the specific service. Security issues of the ports and services is explained in chapter 6.3.4.

OpenBSD 3.6			
Port Number	Type	Service	Explanation
13	TCP	daytime	Returns the local time
22	TCP	sshd	Secure SHell
37	TCP	time	Synchronize time
113	TCP	identd	User and process information

Table 6.3: Open ports for the OpenBSD 3.6 distribution.

6.3.2 Security Risks (OpenBSD 3.6)

Name: auth (identd)

Risk: Medium

Nessus Output: "The 'ident' service provides sensitive information to potential attackers. It mainly says which accounts are running which services. This helps attackers to focus on valuable services (those owned by root)."

Explanation: Report what accounts are running which services, giving possible attackers information, which could be used to attack the system.

Name: sshd, version 1.33 and 1.5

Risk: Medium

Nessus Output: "These protocols are not completely cryptographically safe so they should not be used."

Explanation: Used to connect to host over network from another computer, with a secure connection (encrypted data transfer). These protocol versions of SSH is vulnerable for buffer overflows that may let an attacker gain root access to the server.

6.3.3 SARA Vulnerability Test

Same ports and services that were found by Nessus were found with SARA for this test, but with less information regarding each service.

Name: sendmail

SARA Output: "EXPN command may provide hacker information"

Explanation: This command used by sendmail is sending information in clear text and not encrypted, which may be sniffed by an attacker.

Name: sendmail

SARA Output: "VRFY command may provide hacker information."

Explanation: This command used by sendmail is sending information in clear text and not encrypted, which may be sniffed by an attacker.

6.3.4 Summary and Discussion

The OpenBSD default installation is said to be secure by default but when *Nessus* scanned the system it found that several services were running that are not actually needed. *Auth*(identd) was running which is a pretty common target for attackers, because it gives away information about which applications users on the system is running. By knowing this information as an attacker, the attacker can choose a tactic by exploiting a certain application that is running on the host (if it is exploitable). This information should not be able to get hold of from outside the system. A *daytime* server was also found on the system, which let other hosts get the local time for the system. This is not a necessary service as default. Also a *time* server was running on the system. This service may be used if there are other hosts on the network and all hosts should have the same time. The other hosts will then connect to the *time* server and set their clock to the same time as the server. This is usually not a necessary service by default. Except from the *auth* service that was running there were only one medium security risk found on the system, which was the *SSH* server. The *SSH* server was running protocol 1 for *SSH* transfer, which is not a totally secure protocol. This is fixed by changing the protocol to number 2. No high security risks were found on the system. This could be just as for the FreeBSD version 5.4, a false positive result regarding the Nessus scripts or it could be a true result were the system has not been used for such a long time, which means that possible vulnerabilities has not yet been found. Same ports that *Nessus* found were open according to SARA, but no security flaws were found on these services. There were two possible security holes in the *sendmail* service that was running on the system. These security holes may be false positive, meaning that the exploits may be fixed in the used version of *sendmail.* No other security holes were found on the system. There was not such a detailed report for the *SARA* scanner compared to what *Nessus* reported.

6.4 Experiment Results

The results of the experiment vary for both systems. The systems do not have exactly the same services installed as default because of the installation programs that are used let the user choose what to install. Nessus found more security flaws on the systems, but SARA found other potential security holes than Nessus. The OpenBSD system was running a lot of different services that are not needed as default. These where *daytime*, *time* and *identd.* Even if these services are not exploitable at this time, they could be if some, not already known, vulnerability is found. These services should also be a choice to install during the installation process as they are not needed. On the FreeBSD system these services were not running as default, which is a lot safer. On both systems SSH version 1 was used, this protocol is not completely cryptographically safe and protocol version number 2 should be used instead. This could be easily switched in the configuration file for the SSH daemon. The *cups-base* service that were exploitable in the FreeBSD 5.3-BETA2 is not necessary if there is no printer used on the system. So in this case removing this service would solve the problem. This exploit was fixed in the later version of the system.

The tiff library that was found as exploitable in the 5.3-BETA2 version was installed with *Emacs* and would not be used if this text editor was not installed. As the system do not have X-Windows installed, the tiff library would not be used anyhow as the tiff library is used to display images with the format *tiff*. The *fetch* utility program that was found as exploitable is more critical as this utility is used for the port archive. The port archive is mainly used to download and compile applications in a simple manner. This was fixed in the later version of FreeBSD but should be updated in an older version of the system. The *fetch* utility program was not exploitable in OpenBSD 3.6 and could have already been fixed in this version of the system.

Some other noticeable security breaches on the OpenBSD system is that it allows root login through SSH as default. This is not that safe even if the connection is encrypted. This could be solved by editing the SSH servers configuration file. Also Sendmail is starting automatically on both systems. Whether it should start automatically should be a choice during the installation process, because some of the parts in Sendmail sends information in clear text. Having an insecure Sendmail daemon running on a system could let spam mails pass through the server. This is not just a local problem but would also create huge amount of traffic through the Internet or local network. The server could also be blacklisted on other mail servers, which would cause the server to be unusable for e-mail delivery. This SMTP relay attacks could be controlled by having secure filtering of e-mails through the server, which is done by pattern matching.

The vulnerabilities that were found in the 5.3-BETA2 release of FreeBSD were not found in version 5.4, which shows that the developers have fixed the security holes.

6.5 Summary

The vulnerability experiments were performed on a relatively old computer, which is described in the first part of this chapter, however this should not impact the results. The first part also gives an explanation about the setup of the system and what tools were used to perform the tests. The test was performed with two vulnerability analysis tools that were described in chapter 4 *Attacks and defense*. The result that these tools reported is structured into tables with the open ports that were found on the systems. The security warnings and exploits that were found are displayed in a simpler manner. The chapter has two parts, one for the FreeBSD tests and one for the OpenBSD tests. Why the tests were not performed on the TrustedBSD/SEBSD system is explained in chapter 7.1 *Project Description*, in more detail. The results of the experiment are described in the end of this chapter. As mentioned earlier, this chapter is a more practical part of the comparison, but still very important and a good way to see differences more easily.

Chapter 7

Discussion

This chapter discusses the results for the project. The authors point of view is given as well as tips for further testing of security features of the systems.

7.1 Project Description

The goal of the project was to see what system of the two selected systems FreeBSD with its add-on TrustedBSD/SEBSD and OpenBSD was most secure with as little configuration as possible. The phrase "Most secure" depends in which environment the system is going to be used. A company with demands for multi-user environments needs higher system security than a single user needs. The configuration that was made to the systems depended on what system was installed. Some basic configurations to the systems were performed, including configuration of the network interface card (NIC), file systems (partitioning), choice of which packages to install. The different systems differ in their installations but the packages that were installed were about the same. The SSH daemon was started as default on all tested systems, not only because it was used during the tests but also as it is a widely used remote connection service, which is good to be tested on each system. Other software that was installed was the Nessus and SARA daemon on both systems, except for

TrustedBSD/SEBSD. SARA had the parser generator Bison as dependencies, which had to be installed. The port collection was installed on all tested systems.

The vulnerability tests were not performed on SEBSD for several reasons. The first reason was that a precompiled kernel with SEBSD was used. This kernel was based on FreeBSD 5.2, which was older than the 5.4 version of FreeBSD that was used for the clean FreeBSD installation. To test an older version of FreeBSD would not give any important information regarding security issues as these possible security flaws were hopefully fixed in the later version. The other reason for not testing SEBSD for vulnerabilities was that the structure of SEBSD would not work with the Nessus scanner as a default. Nessus performs tests on the system that checks for known issues with scripts and Nessus needs control of a lot of different services and files. To allow Nessus to have this access in SEBSD the system administrator must implement specific rule sets for the Nessus daemon, else the daemon would not be able to perform the required tests. Allowing the daemon to do all these tests would be like a regular test performed on a system that does not uses these rule sets. The point is that even if the SEBSD system was based on FreeBSD 5.4 the results would be exact the same as for the regular FreeBSD installation if the rule set implementation allowed Nessus daemon to perform all the required tests. So instead of running these vulnerability tests on the SEBSD installation the system was studied to see what different features it had and when it could be a good idea to apply such a system.

7.2 Project Results

As a reader you have hopefully already noticed that SEBSD is a strict system that lets the administrator get much better control of what is permitted in the system. As all programs, i.e. Services, in the system have a rule set, each explains what is allowed or not allowed to do, the programs used on the system are very controlled. The system administrator sets the programs domain or domains and what transitions are allowed for them, if any. These transitions act between domains which let the program change domain from one to another. The administrator can even allow a program to send data through TCP/IP but disallow it to receive data through TCP/IP, and there are a lot of similar control mechanisms.

There are a lot of different positive features that this kernel add-on brings to the system. If a user downloads malicious software or get it by e-mail, the program cannot execute on the system if it has not gained any needed rule sets from the system administrator and it might not even be able to get it downloaded in the first place. If the system administrator installs a software that he think can be trusted but the software contains a backdoor, the system administrator can set the rule sets so the program cannot go over to another domain and make any potential damage, or perhaps not even be able to make the backdoor open for intruders. The rule sets can act as a firewall where the software can be allowed or disallowed to use the network, or parts of the network. There are also rules against misbehaviour on the file systems, which may be good to prevent software to read or write to files that the software should not be able to read or write to. The same rules that may be set for software may be set for users in the system in the form of roles. With the use of the policies that the SEBSD system applies, the system is more secure against new exploits, which also is very positive as updates may not be so necessary to be applied fast.

As the SEBSD system is built up by domains, roles and transitions there is actually a pretty basic concept behind the SEBSD system. But there is a lot to configure before it works as it should. Every program that is to be used in the system needs its own rule sets to work. This means that all new software that is added to the system needs its own rules. It is easier to add new users to the system once the roles have been created as well as the domains. SEBSD has a default scheme for the rules but it is far from complete as every system has different setup regarding software and users.

The SEBSD system with its strict policy rules are securing against threats like worms, Trojans, backdoors and viruses as well as users that may not be trusted. The time it takes to configure the system is long but could be a fair trade for a more secure system. Once the system is configured and up running it is a stable system but if it is going to be a system that constantly needs updates and new software but do not have that many users, than it is not a fair trade as it is taking too much time to create new rules for the applications and may cause new problems. For a company server it is a very good choice for an operating system if the system is going to handle many users, as SEBSD has multilevel security with domains and roles that may protect user's data from other users and possible intruders.

OpenBSD is said to be the most secure BSD system in the default installation. Some issues that were found during the tests and analysis of the system was that OpenBSD allows root login through SSH as a default. This is usually seen as a high security breach and this is something that FreeBSD does not allow as a default. Instead of disallowing root login through SSH there is a message at login that tells the user to use the 'su' command instead, but to be a default secure system, there is not anything positive by letting root login through SSH. It is a lot more secure to use 'su' when already logged in with another user first. An attacker may listen, i.e. sniffing, to the traffic and when the attacker sees a TCP connection he can see that the first data that are sent is the login information and if this information is the root login it could be a lot easier to crack. If the login is performed through another user and if this user is using *su* inside its shell it is a lot harder for the attacker to parse out the correct data from the data stream, which is encrypted.

On all three systems the sendmail daemon was running as a default. As there is not any exploits for this service at the moment the vulnerability tests did not react to this service. But having this daemon running may be a security risk if the system is not continuously updated and someone finds a bug that could be exploited in the sendmail daemon. In OpenBSD there are a lot of system reports and events reported through sendmail to the administrator (root user) account. FreeBSD does not have the same reporting through local e-mail but still the sendmail daemon is running as a default.

What makes the biggest difference between OpenBSD and FreeBSD is that FreeBSD adds the MAC framework which enables ACL and policy models to be used. Without the

MAC framework, FreeBSD basically uses the same access control mechanisms as OpenBSD does. But when it comes to cryptography FreeBSD is more restricted due to the US law, OpenBSD can use different cryptography algorithms more freely. Beyond that there are not that many differences between the systems as both has support for authentication methods and secure connections through networks as well IPSec. The cryptography that was used in both systems are rather weak as many of the algorithms used has been recently broken or detected as weak. OpenBSD has other more algorithms included to choose from than what FreeBSD has. Some of these algorithms are at the moment safer than the default algorithms.

7.3 Author's Point of View

This project has analysed and evaluated two different BSD distributions and they have different features that make them more or less secure compared to each other. Even if one of the systems is less secure in some areas the same system can be more suitable in some environments than a system that does not lack this specific security feature. FreeBSD with SEBSD can be a very secure system if it is configured correctly, but this takes a lot of efforts for the system administrator. Every time a new application is installed on the system, a new rule set needs to be set for the specific application. This is a lot of administration if the system should act as a simple FTP or HTTP server, which many private users are using the systems for. In this case a system like FreeBSD (without TrustedBSD/SEBSD) or OpenBSD is a better choice, which is easier to administrate and does not take up to much time to configure. The security that TrustedBSD/SEBSD offers is very high and its access control features should be used in a system that has many users, i.e. Multi-user environment. For a company server this could be a good alternative if the system are acting as a multiuser environment server. Company systems usually have the same applications running for longer time and no needs for new software, except updates of the current software.

Even if a system like OpenBSD and FreeBSD lacks many of the features that TrustedBSD/SEBSD framework adds they are not unsecure systems. But if a new exploit is discovered, for example in the sendmail daemon, the OpenBSD and FreeBSD systems are a lot more open for attacks than a FreeBSD system with the framework, which may stop this attack from happening as the rule sets reject the malicious operations from sendmail.

These two systems are actually a lot different even if they are siblings from the same system, NetBSD. FreeBSD has a lot easier installation program as well as configuation utility of the system than OpenBSD. This may be thought of something that has nothing to do with security, but it actually does. The simple installation program makes it easy for the system administrator to install the software needed without having too many programs installed that are not used. By only installing the needed software there is less risk to accidently install malicious software. The easy configuration program that is intergrated into the installation program for FreeBSD makes it easy to configure the services needed and to only start these services. In OpenBSD there is not much of a installation program, which makes the unexperienced system administrator helpless and there are a lot of security relevant issues that the administrator may miss. I think that the services that are started as a default on the systems should be asked for before they are started at installation point. There is no need for sendmail in FreeBSD and logging and security warnings in OpenBSD could be addressed to log files instead. TrustedBSD/SEBSD system could be useful for private users as well, e.g. if the system is expected to run for several years and without downtime and no more software should be installed, for example an FTP server. If the rule sets are configured correctly there are not many ways to exploit the services or the server if no new software is installed (that may contain backdoors or Trojans). As I see it, this is a way to put up a server without having nightmares. But I would also consider to use OpenBSD with a good configuration if the server should be a light weight server. OpenBSD claims to be secure by default and the most secure BSD system, but as I see it, a FreeBSD system could be just as secure as OpenBSD if all services used are using cryptography. These services could be the sshd, ftpd and mysqld daemons as examples. There are some features of OpenBSD that speak against the "secure by default" policy, like allowing root login as a default through SSH. The kernel of OpenBSD seems very secure as the developers have heavy auditing of the source code. But still there could be flaws in the code. The FreeBSD project does not have the same way of auditing the code, but many users of the system send in bug reports. As these users may use the bug to exploit systems instead of reporting it there is a high threat against the system compared to OpenBSD where the developers hopefully find the bugs first.

RSBAC could be an alternative for SEBSD. This would be the case if RSBAC was ported to the BSD distributions, which I think would not be that hard to do. As RSBAC adds great freedom of choice of security policy models, it is like later versions of SELinux more open for additional models and changes without reinstalling a system from scratch. An issue with RSBAC would be the verification of security for the protection mechanism itself as it has not that many users, compared to SELinux. SELinux has many users who confirm the implementation of the protection mechanism. OpenBSD does not support any policy mechanisms for the file system. The FreeBSD MAC framework could be used without the SEBSD add-on and could have some more lightweight security policy modules. What also should be mentioned as a security fix is to disable all services that have been enabled in inetd.conf, which is located in */etc/*. When a service is needed it is better to activate this service again instead of having many services running that are not being used.

Even if the security level is very high in a system there are other ways to get hold of the information on a system. And even if there is strict access control for the hard disk drives the information could be gathered by removing the storage device (usually hard disk drive) to another system without this form of security. The information of the hard disk drive is than viewable by the intruder. This is why storage encryption is a good alternative for a system where the importance of integrity is vital. As the data and system information is usually separated on different storage devices the data information could easily be gathered from another system once the storage device is installed. Some security systems have trusted devices which may not be accessed even if the device is connected to another system. The algorithms for encryption in both systems, especially FreeBSD, should be changed to more secure ones or at least use keys with 128 bit or more. Blowfish and AES seem like the best alternative to use or RIPEMD-160 which yet not have been broken. OpenBSD has more choices and are not restricted to US law as FreeBSD is. Regarding cryptography OpenBSD is more safe than FreeBSD.

7.4 Further Investigation

The project has tested the selected BSD systems with vulnerability analysis tools and some basic knowledge about UNIX system security, which includes configuration file analysis and testing of system services. To get a more in depth investigation of the systems, it would be a good idea to perform analysis of the implementation of the different security features in the systems. Specification and verification techniques for the different distributions could be performed as postulated by the Common Criteria that was described in chapter 2.8. But these techniques are very complex to perform on larger systems. Another way to get a more complete study of the security features could be to setup a system and let a lot of different users test it. This would get a more reality based result where users interact in the system. As the tests in this project have been performed with vulnerability analysis tools there could be a lot of different security issues that these tools have missed. The used tools are only testing already known exploitable issues on the systems. A wide range of security analysis tools exists but the frequently updated ones usually costs money, which was an obstacle for this project. An example of such tools is ISS[44] that could be used for further testing of UNIX system security.

7.5 Conclusions

A main conclusion of the project is that TrustedBSD combined with SEBSD is most suitable in a multi-user environment. A multi-user environment on a UNIX system is usually deployed in an organization where role based access control could be used to implement good security policy for the users. Even if the policy models in TrustedBSD and SEBSD aim towards a multi-user environment, they could also be used on a server without many users for preventing attacks from exploited services. For a workstation, FreeBSD is the best choice as it has configuration tools that let the user of the system configure it easily, which leads to better security. It also has tools included to let the user update applications, which leads to up to date software and hopefully better security in the system. If the system is going to act as a server without many users, the choice of system depends on what are the requirements on the security functionality. As example if strong cryptography in the system is important, OpenBSD is a good choice. A FreeBSD system without the TrustedBSD framework could be used with access control lists to get a better access control functionality if the system is having a few users.

References

[1] *FreeBSD (http://www.freebsd.org).*

[2] *OpenBSD (http://www.openbsd.org).*

[3] *TrustedBSD (http://www.trustedbsd.org).*

[4] Charlisle Adams and Steve Lloyd. *Understanding PKI, 2nd Edition.* Addison-Wesley, 2003.

[5] National Security Agency. *http://www.nsa.gov.* Developers of SELinux.

[6] Eva Ansell. *Windows XP.* Docendo Sverige AB, Sverige, 2005-01.

[7] Steve Burnett and Stephen Paine. *RSA Security's Official Guide to Cryptography.* McGraw-Hill Osborne Media, 2001.

[8] Robert Watson Chris Vance. *Security Enhanced BSD.* Network Associates Laboratories, 15204 Omega Drive, Suit 300 Rockville, MD 20850, July 9, 2003.

[9] Advanced Research Corporation. *Security Auditor's Research Assistant.*

[10] Bryan Costales and Eric Allman. *Sendmail.* O'Reilly and Associates, Great Britain, 2003-01-10.

[11] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer-Verlag, 2002.

[12] Alexandre Dulaunoy. *ssh-smart http://www.thinkingsecure.com/ssh-smart.*

[13] Niels Ferguson and Bruce Schneier. *Practical Cryptography.* Wiley, 2003.

[14] GNU Project Free Software Foundation. *awk - data formatting utility, http://www.gnu.org/software/gawk/gawk.html.*

[15] GNU Project Free Software Foundation. *sed - stream editor http://www.gnu.org/software/sed/manual/sed.html.*

[16] GNU Project Free Software Foundation. *wc - word counter http://www.opengroup.org/onlinepubs/007908799/xcu/wc.html.*

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

[18] Jason Garman. *Kerberos - The Definitive Guide.* O'Reilly and Associates, Great Britain, 2003-09-16.

[19] Ben Hammond. *Digital Signatures.* Osborne/McGraw-Hill, 2002.

[20] Duane Ireland and Robin Williams. *MAC OS X 10.4 Tiger.* Peachpit Press Publications, USA, 2005-05.

[21] Ph.D. J. Craig Lowery. *Computer System Security.* Sebastapol, 2002.

[22] Flux Advanced Security Kernel. *http://www.cs.utah.edu/flux/fluke/html/flask.html.* Stephen Smalley, National Security Agency.

[23] Paul Love, Jeremy C. Reed Joe Merlino, Craig Zimmerman, and Paul Weinstein. *Beginning UNIX.* John Wiley and Sons LTD, USA, 2005-05-10.

[24] L. J. L. Padula M. D. Abrams, K. W. Eggers and I. M. Olson. *A Generalized Framework for Access Control: An Informal Description.* Oct, 1990.

[25] Demarc Dynamic Threat Management. *Sentarus (http://www.demarc.com/).*

[26] Bill McCarty. *SELinux.* O'Reilly and Associates, Great Britain, 2004-10-26.

[27] SUN Microsystems. *Pluggable Authentication Modules.* http://www.sun.com/software/solaris/pam/.

[28] Linux Security Modules. *http://lsm.immunix.org/.* LSM Project, Great Britain.

[29] Linda Mui and Eric Pearce. *X Windows System Administrator's Guide, Vol 8.* O'Reilly, 1992.

[30] Christopher Negus. *Linux Bible.* John Wiley and Sons LTD, USA, 2005-02-18.

[31] Luke O'Connor. *On the entropy of arcfour keys.* IBM T.J. Watson Research Center, 1998.

[32] Angela Orebaugh, Simon Biles, and Jacob Babbin. *Snort Cookbook.* O'Reilly and Associates, Great Britain, 2005-04-12.

[33] Common Criteria Project Sponsoring Organisations. *Common Criteria - An Introduction.* Syntegra.

[34] Amon Ott and Simone Fischer-Hbner. *The Rule Set Based Access Control (RSBAC) Framework for Linux.* Karlstad University, Sweden, 2001.

[35] Charles P. Pfleeger. *Security In Computing (2:nd Edition).* Prentice Hall, Inc, 1997.

[36] Sabrina de Capitani di Vimercati Pierangela Samarati. *Access Control: Policies, Models and Mechanisms.* University of Milano and University of Brescia, Italy, 2000.

[37] FreeBSD Project. *Manual for Biba integrity model in FreeBSD.* Nov 18, 2002.

[38] rlogin remote login. $http://linuxcommand.org/man_pages/rlogin1.html$.

[39] rsh remote shell. $http://linuxcommand.org/man_pages/rsh1.html$.

[40] Jerome H. Saltzer and Michael D. Schroeder. *The Protection of Information in Computer Systems.* University of Virginia, Department of Computer Science, Great Britain, 1975.

[41] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition.* Wiley, 1995.

[42] Tenable Network SecurityTM. *Nessus (http://www.nessus.org/about/).*

[43] Gagne Silberschatz, Galvin. *Operating System Concepts.* John Wiley and Sons, Inc., New York, 2003.

[44] Internet Security Systems. *ISS vulnerability scanner (http://www.iss.net).*

[45] Messier Matt Viega Jon and Chandra Pravir. *Network Security with OpenSSL.* O'Reilly and Associates, 2002-06-27.

[46] Webopedia. *Intrusion Detection System (http://www.webopedia.com).*

[47] Naganand Doraswamy Woburn. *IPSec.* PRENTICE-HALL, 2003-04-15.

[48] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. *ssh - OpenSSH SSH client.* 1999.

Appendix A

Wordlist

ACL - Access Control List

BSD - Berkeley Software Distribution

DAC - Discretionary Access Control

DES - Data Encryption Standard (cryptography, NIST, IBM)

DFS - Distributed File System

DSA - Digital Signature Algorithm (cryptography, NIST)

Ext - Extended file system (Linux)

Ext2 - Extended file system version 2(Linux)

FTP - File Transfer Protocol (Internet, RFC 959)

GPL - General Public Licence (GNU)

HTTP - HyperText Transfer Protocol (WWW, RFC 2068)

HTTPD - HyperText Transfer Protocol DAEMON (WWW, HTTP)

LSM - Linux Security Module

MAC - Mandatory Access Control

MLS - Multi Level Security

NFS - Network File System (Sun, Unix, RFC 1094/1813/3010)

PAM - Pluggable Authentication Module (Linux, LISA)

PGP - Pretty Good Privacy

QoS - Quality Of Service, Deliver a guaranteed bandwidth

RSA - Rivest, Shamir and Adleman (cryptography, RSA)

UFS - UNIX File System

UFS2 - UNIX File System 2