Building Secure Distributed Service Solutions

A descriptive evaluation

Master Thesis at Clavister AB, Full paper

Kalle Hagström
Department of Computing Science
Umeå University, SE-901 87 Umeå, Sweden
e-mail: c99khm@cs.umu.se
phone: 070-314 09 30
URL: *http://www.cs.umu.se/~c99khm*

Abstract

Secure critical network products, such as firewalls, are exposed to a large and increasing amount of data and the number of new protocols at application level leads to more complex firewalls. High pressure and complex firewalls are not preferable. One way to handle the problem is to distribute services that operate on application level to a secondary product, an utility server. In this thesis a description of design requirements for secure bastion hosts and a test implementation based on an open-source operating system will be evaluated. The results show that with a secure design for the utility server from network location, hardware to software, better performance for the firewall and a higher degree of security is reached for the system.

Acknowledgements

- Clavister - To the company and all employees for help and the feeling of being one in the team.
- John Vestberg, Clavister - For the original idea, theoretical and practical design.
- Patrik Fors, Clavister - For helping me a lot with operating systems and instant support.
- Fredrik Georgsson, Department for Computing Science Umeå University - For fast and serious support.

Contents

List of Figures

1 Introduction

Computer security is a wide area ranging from human computer interaction to bit level operations in encryption algorithms. One part of computer security is to protect the network resources from malicious people. This is not an easy task, because it covers, just as stated in the beginning, the whole computer security spectrum. The easiest way to protect the network is to unplug it from the outside world, but that solution is useless in a service oriented network. So the dilemma is to offer service and connectivity to network, but only for authorised people.
When building secure bastion hosts such as firewall solutions, it is important to secure each part of the system. If the system will be extended to manage more and more functionalities, there are a high risk for compromising security. In these days where protocols and new techniques are developed continuously, it is even more important to test and secure the new parts of the system when functionality is built into the system. Services for the new functionality can be implemented in many ways, included in the firewall kernel, partial included in the firewall kernel or as a stand alone server. There are advantages and disadvantages with all models. Putting to much functionality into the kernel may lead to kernel bloats ([Sch00]) and dedicate each function a server may produce a complex systems.
In this thesis guidelines how to build bastion hosts will be studied. A test implementation will be made according to the guidelines .

1.1 Background

This project is a part of a master thesis at Department of Computing Science at Umeå University, executed at Clavister AB in Örnsköldsvik. Clavister AB is an *information technology* (IT) security company focused on firewalls and *virtual private networks* (VPN). The products are developed totally by Clavister R&D in Örnsköldsvik. Clavister firewalls use a technology called *Autonomous Firewalling*, this leads to small, secure and stable products. The main issues regarding Autonomous Firewalling Technology is to be independent of any underlying hardware platform and operating system. The firewall is not based on any traditional operating system, such as Windows,

Berkeley Software Design (BSD) or Linux. On the contrary, Clavister firewall consists of a minimal boot loader and a compact firewall core of some hundred kilobytes in size, which constitutes the entire software needed for a complete firewall system, see Figure 1. This means that inherited security vulnerabilities from an underlying operating system are completely avoided. There are great opportunities with this approach, but to keep the firewall small, effective and secure, add-ons have to be external. One add-on is filtering/forwarding of application layer protocols, such as *HyperText Transfer Protocol* (HTTP), *Simple Mail Transfer Protocol* (SMTP), *Session Initiated Protocol* (SIP) and H.323. To keep the Clavister firewall design clean, a stand alone test implementation server will be implemented.

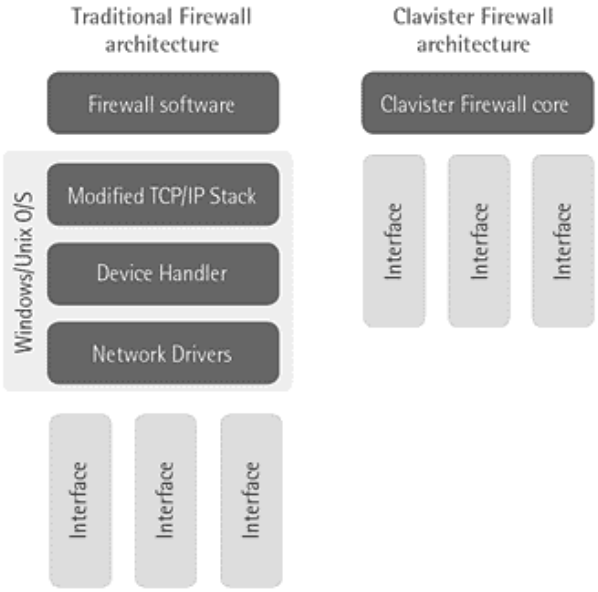

Figure 1: Difference between Clavister firewall architecture and traditional firewall architectures.

1.2 Project Goal

The goal with the project is, from general computer security theory down to specific security problems, to implement and make the right choices for

manage application layer protocol services in a firewall solution.

1.3 Project Specification

The specification is based on the general requirements from Clavister and modified by the author. A more detailed specification can be found in appendix **??**.

1.3.1 Scientific Study

The scientific study will cover general computer security and some examples of security algorithms and implementations. An analytical study over requirements for building secure bastion hosts will be delivered. The study will cover problems and advantages when there is a high degree of functionality in a product/program. Security problems when building secure systems and firewall systems will be in focus in the report. The implementation will be a solution or one way to attack the problem. The idea of the implementation will not be Clavister specific. The report should give guidelines and recommendations for building a secure utility server. The implementation must also be motivated according to scientific literature.

1.3.2 Practical Study

In the practical study a dedicated utility server running services for managing application layer protocols, such as HTTP, SMTP, SIP and H.323, will be implemented. The utility server will be a stand alone server connected to the firewall via a dedicated Ethernet interface. The operating system should be based on an open-source operating system, like Linux, BSD or similar. One goal is to present why a certain operating system was chosen in favour of others. The run-time environment should be kept to a minimimum. As many of the standard binaries as possible should be removed. The entire base environment should fit on a small sized flash media. The design of the run-time environment should be designed in a simple way to facilitate upgrades. The utility server should be equipped with a boot menu and it should be accessible through serial port (RS232) or from the VGA/Keyboard console. The boot menu should be responsible for initial settings like *Internet Protocol*

(IP) address, netmask, gateway and system related tasks. Boot scripts must be adjustable for the system to provide a secure boot sequence.
Service components that might be appropriate for the utility server are HTTP proxy (*Uniform Resource Locator* (URL) filtering, Web Cache), mail relay (SMTP), secure *Domain Name System* (DNS) service, SIP proxy and H.323 proxy. Each service should be able to be configured from the Clavister Firewall Manager.

1.4 Constraints

The theoretical study will cover general computer security and requirements for building secure bastion hosts. Firewalls will not be described in detail, only a brief explanation will be given.
There will only be one practical type of implementation of the utility server, but different flavours of the type will be tested. The services running on the utility server will not be tested, focus will be on the base system.

1.5 Method

The project will be divided in different phases. All phases will be in corporation with Clavister. The first phase is the planning phase. It will be discussed with the supervisor in order to decide milestones. Constraints and a more specific technical specification will be planned. The research phase and the report writing phase will be combined and written in decided style and formats. The implementation phase will be executed together with Clavister at the Clavister office.

2 Security

Security is a wide area and must be defined for each segment, since it covers from door locks to cryptography. For computer systems, security is to a high degree about protection. Protection might be everything from physical protection of a server to confidentiality. To achieve security, requirements are defined to protect and distribute data. These requirements are issues as confidentiality and authorisation. It is important to make distinctions between requirements, theory, and technologies. A requirement is used to define a goal, for example *this data should not be read by a third part.* Encryption theory is used to define new technologies as *Secure Socket Layer* (SSL).

2.1 General Security Requirements

In all distributed systems there are transmission requirements for sending messages. To achieve these requirements encryption and hashing functions are used. The requirements can be divided into five categories.

Confidentiality - Make sure message cannot be read by a third part.
Example technique: *Secure Socket Layer* (SSL)
Authorisation - Control that the sender is granted to send the message.
Example technique: Access control lists and policy implementations.
Data integrity - Ensure that the message remain unchanged between sender and receiver.
Example technique: Hashing functions, example *Message Digest Algorithm 5* (MD5).
Message origin authentication - Make sure the message is not a replay.
Example technique: Hashing functions with nonce values and secret keys, example *Keyed-Hash Message Authentication Code* (HMAC).
Non-repudiation - Indemnify the originator of the message cannot deny its origin.
Example technique: Digital signatures ([Sur01]).

2.2 General Threats and Attacks

The threats are usually categorised into the following subareas.
Leakage - Messages are spread to unauthorised recipients.
Tampering - Messages are changed without authorisation.
Vandalism - Obstruction of a correct system operation.

The attacks are normally classified in these five categories.
Eavesdropping - Obtaining message copies without authority.
Masquerading - Using an identity of another principal without authority.
Message tampering - Intercepting and altering messages.
Replaying - Storing messages and send them later.
Denial of service - Sending a large set of messages to a resource or on a channel to deny access for others ([Kin01]).

2.3 Security Theory and Technology

When implementing security in applications, theory such as encryption and hashing functions are used. Those theories are often combined to new technologies to achieve requirements stated in 2.1. Encryption is used to communicate in a secret way or to keep information secret to your self. Hashing and message digest functions are used to ensure data integrity, non-repudiation and message origin authentication.

2.3.1 Symmetric Key Encryption

Symmetric key encryption ([Ros01]) is also known as shared secret key encryption ([Kin01]). To use this encryption method both parts, here called Alice and Bob, have to share the same key.

1. Alice uses the secret shared key K_{AB} and encrypts the message with a agreed encryption algorithm $E(K_{AB},M)$ and sends the encrypted message $M_{K\,AB}$ to Bob.

2. Bob decrypts $D(K_{AB},M)$ the message with the shared secret key K_{AB}.

The advantage with this type of encryption method is speed and simplicity. The disadvantages are:

How can both parts send a shared key securely?
How does the receiver know that the message is not a replay?

The most known symmetric encryption algorithm, except from basic ones as Caesar ciphers, are *Data Encryption Standard* (DES), 3DES, *Advanced Encryption Standard* (AES), *Ron's Code 4* (RC4), RC5, *International Data Encryption Algorithm* (IDEA) and Blowfish ([Sch00]). DES, data encryption standard, was published in 1977 by the *U.S. National Bureau of Standards*. The effective key length is 56 bits long, which is to short nowadays. In 1999 a team called *Electronic Frontier Foundation* (EFF) cracked DES in 22 hours and 15 minutes in a distributed attack ([Ros01]). To provide a higher grade of security with DES using a 56 bit long key, triple-DES (3DES) was introduced. 3DES simply run DES three times, but this of course reduce the speed of the algorithm.
Because of the lack of an effective and secure symmetric algorithm, U.S. *National Institute for Standards and Technology* (NIST) introduced a competition in 1997 to adopt a new standard algorithm, *Advanced Encryption Standard)* (AES). In May 2000 an algorithm called Rijndael was chosen ([CSR02]). AES uses can use 128, 192 and 256 bits key length. If someone could build a machine that cracks a DES-encrypted message in one second, it would take 149 thousand-billion years to crack 128 bits AES crypto with the same machine. The difference in key length is the major factor. Adding one bit to the DES key will double the time to two seconds and with a difference by 72 bits the time will be 2^{72} times longer.

2.3.2 Asymmetric Key Encryption

Public key or asymmetric key encryption is used to avoid sending secret keys to each other. Asymmetric key encryption is a fairly new method. It was first published by Whitfield Diffie and Martin Hellman in 1976, but it was probably already discovered by the British intelligence service a few years earlier ([Sch00],[Ell87]).
With the following notation:
$M = Message$

E = *Encryption algorithm*
D = *Decryption algorithm*
K = *Key*
the theory of asymmetric encryption could be described as

$$D(K_{private}, E(K_{public}, M)) = M \text{ ([Kin01])}.$$

The greatest advantage with asymmetric key encryption is, as mentioned before, lack of distribution problems as with shared secret keys. With this method each participant has one secret key and one public key. When for example Alice wants to send an encrypted message to Bob, she encrypts the message with Bob's public key and sends it to Bob. When Bob receives the message he decrypts the message with his private key. The basic idea to achieve these properties are functions that are easy to compute one way, but almost impossible in the other direction. Such function can be integer factorisation for prime numbers. It is easy to multiply two large ($>100^{100}$) prime numbers, but finding the factors of that product is almost impossible. The most known algorithm for public-key schemes is *RSA*, named after its founders Ron Rivest, Adi Shamir and Leonard Adleman.([Ros01])
The first problem with asymmetric key encryption is the uncertainty to trust that the right public key is used. Therefore is it very important to publish the public key at trusted authorities. The second problem is speed. It is not realistic to use asymmetric key encryption in ordinary secure communication. Another better way is to use hybrid cryptographic protocols, which use asymmetric key encryption to exchange a shared secret key. This is described in 2.3.4.

2.3.3 Hash Functions

Hash functions or one-way hash functions are functions only computable one way. This makes them very usable in computer security, because of the property to make small-sized digital fingerprints of large-sized data. Hash functions do not use any keys, it is only a mathematical function that gives a fixed sized answer called message digests. The properties of hash functions must be very strict.

- Given message M it should be easy to calculate message digest x.
- Given message digest x it should be hard to calculate M.
- Given a message digest x for a message M, calculated as $H(M) = x$ it should be impossible to find another message $M2$, such that $H(M2) = x$.

Hash functions are used in many applications, especially when it comes to non-repudiation problems. The most known one-way hash functions are *Message Digest 5* (MD5) and *Secure Hash Algorithm 1* (SHA1) ([Kin01],[Ros01],[Sch00]).

2.3.4 Confidentiality, Secure Socket Layer

Secure Socket Layer is a technique which provides an encrypted channel between a client and a server. The motivation to use it is protection of the data between sender and receiver. It was originally developed by Netscape Corporation and became a *de facto* standard in secure connections on the Internet ([Kin01]). A newer protocol called *Transport Layer Security* (TLS) has been adopted which is described in *Requests For Comments* (RFC) 2246 ([Tre02]).
SSL consists of a hybrid scheme, which means it uses clear text to set up the connection, asymmetric encryption to exchange the shared private key and finally symmetric encryption. Before the encryption starts the client and the server negotiate what algorithm to use. SSL is two layered with four different protocols as in Table 1.
To set up a SSL connection the client uses the *SSL Handshake Protocol*. In the first phase the client sends a message in clear text to the server. The message contains data about the clients algorithms to encrypt and verify messages. The server also replies with a not encrypted message. This message contains the server's digital certificate and algorithms to encrypt and verify messages. When the client receives the digital certificate from the server, the client verifies whether the certificate is correct or not. The criteria for a correct certificate are a correct time stamp, a trusted *Certificate Authority* (CA) and the correct domain name. The domain name must be connected

to the certificate. This part of the exchange of data is exposed for *man-in-the-middle* attacks, which is a form of message tampering described in 2.2. To avoid these attacks the public key is often delivered in separate channels, such as CD-ROM. The other protection is the domain name in the certificate. It is only recommended to exchange data with the same IP address corresponding to the domain specified in the certificate ([Kin01]). If the certificate is correct, the client chooses algorithms according to the cipher suite the server offered. Optionally the client may choose a compression algorithm, but it is not needed for a secure session. In this stage the client creates a pre-master secret, which is later used to generate a shared private session key. The pre-master secret contains the chosen algorithms, the information which is already exchanged, and a large random value. Before sending the information to the server, it is encrypted with the server's public key. The server decrypts the message with its private key and generates a shared private session key. It is now possible to start a secure connection. Both parts have equivalent session keys and they know the type of encryption to use.
The *SSL Record Protocol* is used to encrypt and authenticating messages. The data flow is described in the following enumeration ([Kin01]).

Data flow from sender in SSL Record Protocol

1. Read the data from the application.
2. Fragment or combine the data to manageable blocks.
3. Optionally compress. This is not needed for a secure connection but it can increase the speed of the encryption and the signing of the message.
4. Message authentication, discussed in Chapter 2.3.7.
5. Encrypt the data.
6. Send as *Transmission Control Protocol* (TCP) packet.

The *SSL Alert Protocol* contains only two bytes. It is used to alarm the counterpart if the received data is defective. *SSL Change Cipher Protocol*

SSL Handshake Protocol	SSL Change Cipher Specification Protocol	SSL Alert Protocol	HTTP	Telnet	...
SSL Record Protocol					
Transport layer (usually TCP or UDP)					
Network layer (usually IP)					

Table 1: SSL protocol stack ([Kin01])

is used to change cipher or type authorisation in an already existing SSL session. This is useful when a session starts only with encrypted channel, but later in the session the messages need to be signed.

2.3.5 Authorisation, Access Control Lists

To provide authorisation to different resources, access control lists can be used. Every resource has an access control list with entries on the form *<domain, resource>* for each domain that have access to the specific resource. This is used in operating systems, such as Windows NT and all Unix flavours, to set access permissions on files and directories. When accessing a file the domain restrictions of the user is compared to the entry in the access control list of the file ([Kin01]).

2.3.6 Data integrity, Message Digest Algorithm 5

One technique to get data integrity is hash functions. *Message Digest Algorithm 5* (MD5) was designed by Ron Rivest in 1992 as an improvement to MD4 and is described in RFC 1321 ([Riv92]. It is an efficient one-way hash function producing a 128-bits long message digest, by applying four non-linear functions to each 32-bit segment in a 512 bits long block of the message ([Sch96]). The functions are:

- $F(X, Y, Z) = XY \vee \neg(X)Z$
- $F(X, Y, Z) = XY \vee \neg(X)Z$

- $G(X,Y,Z) = XZ \vee Y\neg(Z)$
- $H(X,Y,Z) = XxorYxorZ$
- $I(X,Y,Z) = Yxor(X \vee \neg(Z))$

MD5-algorithm

1. Append padding bits. A single set bit is added to the message M followed by zeros until the length is congruent to 448, modulo 512. (e.g. *length(M) mod 512 = 448*)

2. Append length. Add a 64-bit representation of length of M to the end of the new message from previous step. Now the message has a length that is exact a multiple of 512.

3. Initialise message digest buffer, as a four-word buffer (A,B,C,D) as follows:

```
A: 01 23 45 67
B: 89 ab cd ef
C: fe dc ba 98
D: 76 54 32 10
```

4. Process message in 16-Word Blocks, mangle the blocks in for rounds. (see ([Riv92]) for further information)

5. Output. The digest contains the buffers A,B,C,D with the low-order byte A.

There have been several attacks to crack the MD5-algorithm, but no one have been successful. Ron Rivest himself claims *"the difficulty of coming up with two messages having the same message digest is on the order of* 2^{64} *operations, and that the difficulty of coming up with any message having a given message digest is on the order of* 2^{128} *operations."* ([Riv92]).

2.3.7 Non-repudiation, Digital signatures

In cases of normal paper documents, handwritten signatures are used to verify a document's source. Signatures are used to prove that a document is authentic, unforgeable and non-repudiable. This is not totally fulfilled with handwritten signatures. It is possible to make copies of a signed document and change some parts.
Digital documents are easy to create, copy and modify. There are no meaning to add a signature like a name, a picture or a handwritten image to prove its origin. The signature can easily be changed and added to other documents. Digital signatures have another approach than handwritten signatures. They are based on cryptography instead of the handwriting pattern. There are two ways to do this, with public keys or with private keys.

Digital signatures with public keys

1. Alice has two keys, one private $K_{private}$ and one public K_{public} which is accessible from a *Public Key Infrastructure* (PKI) service.

2. Alice computes a hash value of the message M, $d = h(M)$ and encrypts it with $K_{private}$ to create the signature $S = E(d)_{Kprivate}$.

3. Alice sends the message M and the signature S to Bob.

4. Bob decrypts the signature with Alices public key $D(S)_{Kpublic}$ and computes the hash value of the message M, $d' = h(M)$. If $d = d'$ the signature is valid.

Digital signatures with private keys, Message Authentication Code (MAC)

1. Alice generates a key K and sends it to Bob in a secure channel.

2. Alice computes a hash value of the message M and the key K, $d = h(M+K)$.

3. Alice sends the message M and the hash value d (d is the MAC) to Bob.

4. Bob adds the shared secret key to the message M and computes the hash value
   $d' = h(M+K)$. If $d = d'$ the signature is valid.

The advantage of using private keys is speed. It is 3-10 times faster than using public keys. On the other hand there are problems with exchanging keys in a secure way. In the example given above it is possible for Bob to sign a message and claim that it was Alice who wrote it ([Kin01]).

2.4 Secure Systems

When talking about security, the techniques and tools, as mentioned above, are often set in focus. The algorithms and techniques are often validated and working properly, but when it comes to bigger systems, bugs and bad implementations can reduce the strongest encryption algorithms to open holes into the system. Cryptography is an isolated island, when the system is the whole sea and therefore a lot harder to secure.
Designing secure systems is a difficult task and it is impossible to claim that a system is totally secure. The biggest problem is to exclude every possible attack, bug, malicious employees etcetera. There are always problems that the designers did not think of or new problems and ways to attack the system. Be prepared to all possible threats are one part of system security, but it is also easy for a system administrator to make mistakes when setting up user accounts, choose the right virus protection or manage databases. To secure a system the designers have to avoid disasters and mishaps. The errorness in a system can be reduced by using formal validation tests. To perform such tests it is a good idea to write down all possible attacks and threats and show how to prevent them in the system. When starting designing a secure system it is always good to assume the worst-case scenario for different problems. Worst-case assumptions and design guidelines are listed as follows in [Kin01].
Interfaces are exposed - In systems where networks are involved interfaces are used to share services or resources and they are therefore open for clients.
Networks are insecure - Message sources can be falsified, host addresses can be spoofed etcetera.

Limit the lifetime and scope of each secret - The longer time secrets, such as passwords and keys, are used the greater risk for the secret to be compromised.
Algorithms and program code are available to attackers - It is better to use public encryption algorithms than secret encryption algorithms. The secret should be in the key not in the algorithm.
Attackers may have access to large resources - In the new networked world it is possible to use a big number of computers in attacks, such as in the attack against Yahoo year 2000 ([Ros01]).
Minimise the trusted base - All hardware and software the system is based on has to be trusted, therefore it is better with a very hardened base.

Different theoretical models to secure systems have been developed during the years. One model, Bell-LaPadula is a model for defining access control in a system, mainly for government and military use. The model defines subjects (persons, other systems) and objects (applications, resources, files). A subject can only access objects on a certain level, defined by its security level.
The model consists of two major rules, one for reading data and one for writing data. Four levels of security is defined, top secret, secret, confidential and unclassified. A subject with the access security level secret is allowed to read secret, confidential and unclassified data, but the subject can only write secret and top secret data. If the subject with the security level secret was allowed to write data at lower level, it would be possible for the subject to down classify data. This could be a problem if for example the mailing rules are set to only allow outgoing mail with unclassified or confidential data.

These access rules are called *mandatory access controls* since they are required by the system. The definition is:
"Restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorisation (e.g., clearance) of subjects to access information of such sensitivity."([Dew96]).
The opposite is discretionary access control, which is used in operating sys-

tems as Unix and Windows NT. In those operating systems it is the subject who defines what resources others can access. The definition is:
"Restricting access to objects based on the identity of subject and/or groups to which they belong." ([Dew96]).
Even if the Bell-LaPadula model works, there are some major disadvantages. It is only focused on confidentiality, which is very useful for military systems, but for normal use other properties are needed. Someone has to classify everything and that is a big problem to manage. Resources can be aggregates and needs different classification. Other models are *The Chinese Wall model* and *Clark-Wilson model* ([Sch00]).

2.5 Security Policy

The main part about security is sharing resources in a limited way ([Kin01]). Everybody should not be able to share resources and everyone should not be allowed to do whatever he/she wants in the system. That is the main reason for adopting security policies, also called trust model ([Sch00]). A security policy defines the limits, the goals and the aims. Examples of possible items for a company's policy might be:

- Do we need a firewall and how should it be configured?
- Do we allow users to download .doc .exe .zip .vbs etcetera?
- Do we allow streaming media?
- Should net A be accessible from net B?
- Which employees should have access to the buildings?
- Do we only need passwords or do we need any access tokens?

For a more detailed security policy see appendix A. A security policy is a wide description which often needs to be divided into sub-policies, such as *Acceptable Use Policy* (appendix A), *Automatically Forwarded Email Policy* and *Remote Access Policy* ([SAN01].
With a good security policy the risks for leaks and security problems are reduced, but it is not enough with a good policy. The most important task

is to make the employees aware of the policy. It is no meaning with a policy telling the employees not to open attached files in e-mails, if no one is aware of the policy. As a system administrator or responsible for security it is a good idea to implement as much automatic controls as possible in the rule set. For example if there is a rule in the policy telling *"do not down load .exe files"*, the firewall should stop those files.

3 Firewalls

As discussed in section 2.5 a company needs a security policy to achieve good security. One part of a security policy should address firewall issues. Firewalls are not well defined, there are many different types and for different use. In general, a firewall is used to provide access to services in an internal network for the external users of the organisation, but stop other users. In [Kin01] it is stated as, *"The purpose of a firewall is to monitor and control all communication into and out of an intra net."*. The security policy for a organisation's firewalls can be listed by the following:

Service control is to decide which services or internal hosts should be accessed by external users. Other users should be rejected by the firewall. Outgoing traffic should also be controlled. The decisions if a request should be rejected or not can be based on the content in the IP packets.
Behaviour control is to prevent the users in the organisation to do such things that might harm the security in the network. It is better to trust the firewall, than the users. One problem is that this often requires to go higher in the stack, often to transport level but sometimes up to application level. One example of behavior control might be spam filtering.
User control is to distinguish the users of an organisation from each other. Everybody should not have the same access rights to resources, internal and external, as others ([Kin01]).
A firewall works as the name intimates, a wall against the outside world. This is the major drawback with firewalls. They do not prevent attacks from the inside of a network ([Kin01]). Often an attacker from the inside can make more harm, than an intruder from the outside. It is more dangerous if the organisation is a school or university than a company. *"We have often observed that there's nothing quite so dangerous as a bored MIT student.* ([Bla00]).
A bad assumption is that the network is secure just because the organisation uses firewalls ([Bla00]). There are often back doors into a network, especially if the organisation does not have a good security policy that the collaborators are aware of. Back doors might for instance be a local modem under the desk. If only one type of web browser is allowed, internal proxies that

translates the request from a prohibited web browser to a permitted can be used. If the firewall is inconvenient to the users, there is a greater risk that users create back doors ([Bla00]).
Todays programs (read Microsoft Office ™) often mix executables and other data. This is a threat to the security and there is no way to stop the integration in the programs with firewalls. The only help a firewall can give is to prevent users from downloading such files (.doc, .xls) from the Internet ([Bla00]).

3.1 Types of Firewalls

Firewalls can be categorised in mainly three types based on the technique and in what level the packets are examined. The most basic firewalls are packet filtering firewalls, which check the packets on network level. The extreme case in the other end is the application gateway firewall, which checks the packets at application level. In the between, stateful inspection firewalls are found. The stateful inspection firewall keeps the state of network connections and examines the packets at multiple levels, from network level to, in some cases, application level.

3.1.1 Packet Filtering Firewalls

This is the simplest form of firewall and therefore the most inexpensive firewalls of today are packet filtering firewalls. The firewall picks up all packets and checking the header according to a set of rules. Normally packet filtering is implemented in a router sometimes called *screening router* ([Cha00]) or in a secure host with two network interfaces ([itw02]). The filtering is made of set of rules typically based on:

- The IP source address (supposedly).
- IP destination address.
- TCP or UDP source and destination port.
- ICMP message type.
- Connection initialisation datagrams using TCP *SYN* or *ACK* bits.

- Packet size ([Ros01], [Cha00]).

When designing the filtering policy for a organisation there are some normal cases to start with. One example is to reject all packet from the outside claiming coming from the inside. This type of packet is often part of a spoofing attack. But unfortunately there is no way to check if a packet from the outside really originates from the IP address it is claiming.
Another basic filtering policy is to prohibit all connections to port 23, which corresponds to Telnet sessions. This prevent outsiders from logging on to internal hosts using Telnet ([Ros01]).
Packet filtering is very effective. It does not reduce the throughput much, but on the other hand it is static and not to secure for bigger organisations. The filtering is only performed on IP packets as in Figure 2. The filtering is often performed in an operating system kernel for performance reasons ([Kin01]).

The packet filtering firewall only examines IP packet, which makes it more or less useless to attacks to applications. As an example, if the filtering policy permit outsiders to use port 80, the web server port, the firewall does not prevent attacks to the web server itself ([itw02]).

3.1.2 Stateful Inspection Firewalls

This type is the most frequently used firewall. Instead of inspecting each packet individually, the firewall keeps information about the state in the network and what types of packets are expected ([Sch00]). This type of firewall is an extension of packet filtering firewalls.
Stateful inspection firewalls gathers information from the incoming packets from one interface to determine the connections state. Example of information might be TCP sequence numbers. If the connection passes the rule set, similar as the one for packet filtering firewalls, the packets are forwarded to another network interface. The information that is gathered from each packet is used to build dynamic state tables. The tables are used to keep

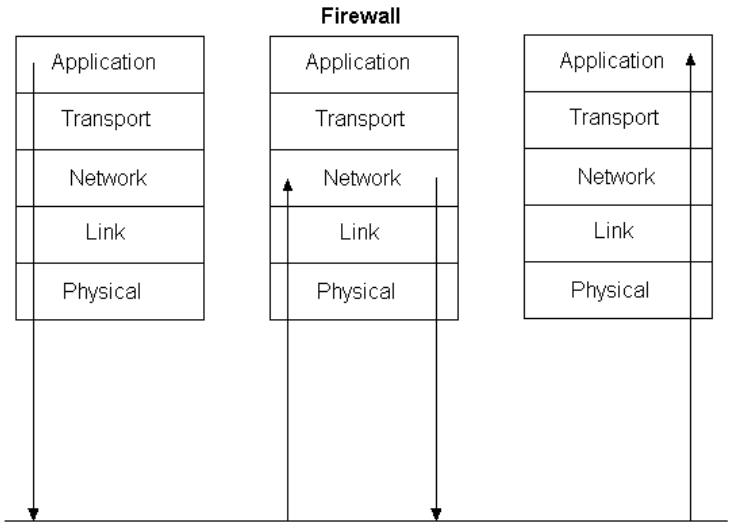

Figure 2: Packet filtering firewall, protocol stack.

track of a connection though the firewall. Instead of evaluating each packet to a rule set as in packet filtering firewalls, the stateful inspection firewall only allows packets that are part of a valid connection to pass ([itw02]). Decisions if a connection is valid or not can be based on different layers in the protocol stack as in Figure 3 ([HKU97]).

The biggest advantages for stateful inspection firewalls are the increased security over packet filtering firewalls (see 3.1.1) and increased performance compared to application gateway firewalls (see 3.1.3). Stateful inspection firewalls only checks the packet at the level that is necessary. Another advantage is the simplicity to add new services to the firewall ([HKU97]). Most of the high-end firewalls are statefull inspection firewalls.

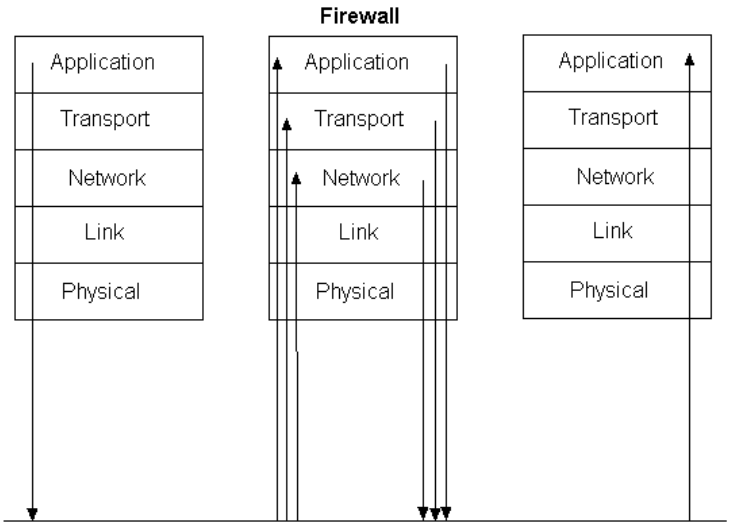

Figure 3: Statefull firewall, protocol stack.

3.1.3 Application Gateway Firewalls

Application gateway firewalls, sometimes called proxy firewalls, inspect the packets at application protocol level , see Figure 4. This type of firewall terminates each connection and redirects the connection if it is valid, according to the rule set. They are therefore often called store-and-forward proxies ([Sch00]). The firewall is transparent between the internal network and the external. All communication is made to the application gateway firewall instead of directly to the parts of the communication. The firewall presents the illusion that the user is communicating directly to the real server and vice versa.

By examining all packets at application level, a higher degree of security is reached, but only if the proxies are well implemented. At this level it is easier to perform intelligent filtering, such as looking at content in the packets and

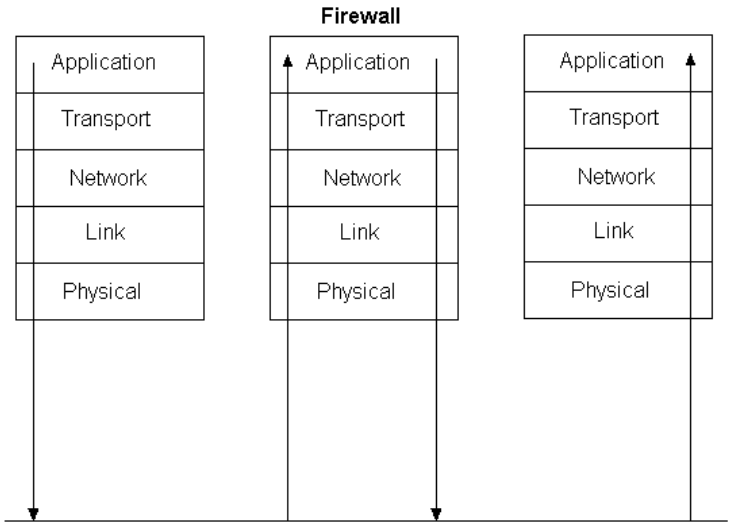

Figure 4: Application gateway firewall, protocol stack.

virus checks. Since the connection is terminated, the firewall generates new IP packets to the client, which automatically leads protection for weak or faulty IP implementations. Another aspect that is not a security advantage, rather network efficiency advantage, is the caching possibilities ([Cha00]).
All these advantages do not come without significant drawbacks. First of all is the latency high and lower throughput because the firewall examines all packets at the highest level in the protocol stack ([Sch00]). Another disadvantage is the problem to get proxies to new services that uses other protocols. It is easy to find proxies for *File Transfer Protocol* (FTP) and Telnet, but when it comes to newer services there are problems to find suitable proxies. The final problem is a consequence of the fact that services needs different proxies. This is because of the proxy needs to understand each protocol to determine if the packet are correct or not. Finally, application level gateway firewalls often requires modifications in clients and applications ([Cha00]).

3.2 Firewall Products

In section 3.1 three different types of firewall architectures were described. Those types can be implemented and manufactured in different ways. The easiest way is to implement the firewall on top of an ordinary operating system, using the IP implementation of the operating system. Another solution is appliance firewalls, which is pre-installed software on selected hardware. Finally hardware firewalls, which use specialised hardware to provide fast throughput.

3.2.1 Software Firewalls

Software firewalls are based on a traditional operating system, such as Linux or Windows NT. This makes them easy to develop. Often an API for system calls already exists. This is the easiest way to develop firewalls.
The major drawback is that the the firewall has to trust the underlying operating system. As will be discussed in section 4.4 in a larger system with more lines of code, the total number of bugs increases. Another problem with a traditional underlying operating system is that it is a tempting target to attack. There are also problems with functionality in the firewall when upgrades of the operating system has to be performed ([Cla01]).

3.2.2 Appliance Firewalls

Appliance firewalls are firewalls that are manufactured as an all-in-one solution. The firewalls are shipped with hardware and software pre-installed ([web02]).
When creating appliance solutions the great advantage is the possibility to have control over all parts in the system. From hardware choices to software implementations. The manufacturer can test the configuration before it is shipped to the customers. This is also a good service for the customers, since they do not need to configure that much. The biggest drawback is the development costs. It is more expensive to test new hardware and the development costs are higher as well.
The software in appliance models might be based on a traditional operating

system, like Linux or Window NT, but there are also possibilities to develop firewall kernels based on company specific operating systems.

3.2.3 Hardware Firewalls

The fastest firewalls are built on specially developed hardware. All computation is made on especially adjusted chips that accelerate the performance. The buses are also often special adjusted for the product. Of course, some type of operating system and firewall kernel is needed. As stated earlier, performance is the biggest advantage for hardware firewalls, but they are extremely expensive to develop. This is easy to understand, because each part of the firewall has to be specialised. Another drawback is keeping up to changes in protocols and the support for new protocols. It takes longer time for hardware firewalls developers to meet new requirements than for software firewalls ([Net01]).

3.3 Services in Firewalls

When designing firewall solutions the location of services, like different proxies, is important. The most simple way is to implement the service in the firewall. In application gateway firewalls where the packets are examined at application level, this might be the best solution because the proxy must check the packets at application level anyway. But with large applications the number of bugs increase ([Hum99]) and a firewall is a critical part of the network solution.
Packet filtering and stateful inspection firewalls, examine the packets in lower layers, like transport and network layer, instead of the application layer. This implies a higher degree of inspection if the service is implemented in the firewall. Then the better performance in stateful inspection and packet filtering firewalls is lost. The security is also reduced, because of the complexity of the system ([Sil01]). In these circumstances it is better to separate the firewall from the services and build a distributed solution. Then the firewall only examines packets at the necessary level and focus on security. The service servers can also be placed at demilitarized zones to secure them more and the solution in each server/firewall is less complex ([Ves02]).

4 Analysis, Building Secure System with Bastion Hosts

Secure bastion hosts are like castles, only granted people are permitted to enter the building, no one else. Inside the castle different types of activities are performed, guarded by guards. A bastion host is typically a firewall or external service hosts. By default bastion hosts are exposed to high risks, because of their location and task. There are two basic principles when designing and building bastion hosts, keep it simple and be prepared to be compromised. These two principles must imbue all parts of the design.

4.1 System Design

Design of systems is not an easy task, when the systems are networked and distributed there are even more challenges. The designers of the system should consider to take the listed characteristic into account.

Heterogeneity

In the networked systems there are huge heterogeneity, like different networks, hardware, programming languages, operating systems and applications. The big challenge is to make all parts cooperate.

Openness

If a system is easy to re-implement and extend it has a high degree of openness. In distributed systems this is important because new parts can be added on other places in the system, that have to use services on other parts. Openness can be achieved if specifications and software interface (API) are well defined and published. This is easy for in-house products, but to new protocols and products it is often a very slow process. New protocols are often published as *Request For Comments* (RFC) to facilitate the process.

Security

Security is much more important in distributed system, than in applications running on a computer not attached to a network. In distributed systems data are transferred and stored at different hosts. These data is often attractive to other users. The security is not only about protecting the data, authorisation and non-repudiation are other security issues. This is discussed in section 2.1.

Scalability

A system is scalable if it is still effective and efficient if the number of user and resources are increasing dramatically. The scalability definition includes cost, performance and software resources. The cost of physical hardware resources should be reasonable if the pressure increases. In general the cost of new resources, such as new servers, should be proportional to the capacity of the original server. For example this can be expressed as when adding one server, the capacity should be doubled if only one server exists from the beginning.

The performance loss must be controlled when the pressure on the system increases. Therefore the algorithms must be designed to use hierarchic structures, rather than linear structures. But there are always performance loss when the data set is increasing.

One example of bad scalability when it comes to prevent software resources to run out, is the design of Internet addresses. There, the upper limit of 32 bits, prevent scalability. Bottlenecks in general must be avoided to achieve scalability.

Failure handling

When one part of the system fails or crash, it must be handled by the rest of the system. First of all the failure must be detected, but that is not as easy as it seems. The problem is to detect where the problems are. If failures occur and are detected they can sometimes be masked or made less severe. Retransmission of data is one example of masking failures.

There will always occur failures in distributed systems. Tolerating failures is also one way to manage some problems. One way to tolerate failures is to inform the user and try later. To deal with permanent data failures, recovery is important. This can be redundant disks or rollback functions in databases.

Concurrency

All resources that are shared in a distributed system must handle concurrency in an appropriate way. In a concurrent environment the objects must be safe according to operations and keep the data consistent.

Transparency

When the user uses the system it should see the system as one part, not as separated parts. There are different types of transparency, like access, location, concurrency, failure and performance transparency. Access trans-

parency makes no difference in local or remote access and local transparency enables resources to be accessed without knowledge of the location ([Kin01]).

The most fundamental principle with secure systems is least privilege. The principle purpose is only grant privileges enough to each object that it needs to perform the task. This can be anything from users read-write privileges to firewalls trusting networks. Programs which runs with root privileges are always dangerous because of the full privileges ([Cha00]).

Simplicity is another strategy for secure systems. Simple systems are easier to understand and less buggy. Complex programs have more bugs both harmful and harmless. Many harmless bugs tend to make the harmful hard to detect. Of course, it is not recommended to sacrifice security to get simplicity ([Cha00]).

4.2 Hardware

When building security products, reliable hardware is very important. Reliable hardware is not the newest hardware, the leading edge technology might be the bleeding edge ([Cha00])! If no *Central Processing Unit* (CPU) consuming application, such as Web server, will run on the machine, a normal desktop machine is enough. But it is better to have a server packed machine, because they are harder to steal, easier to change hardware (PCI hot-swap) and have less risk to be turned off.
The best choice is to have no movable parts (hard drives) on the machine, but this might be impossible if a caching proxy will run as a service because caching proxies usually need a lot of memory. If the machine is really critical, redundant disks and uninterruptible power is to prefer.

The speed of the machine is usually not a bottleneck. It does not take much CPU power to handle proxy services. There are several other reasons not to choose the fastest machine. It is less prestige to compromise a slower machine and if it is compromised a slower machine is not so useful. It is not a good idea to use a slow machine for brute-force and dictionary attacks. That is from the outside. Users on the internal net might find a fast machine as a good compilation server or database server. This will make maintenance of security a difficult task ([Cha00]).

4.3 Physical and Network location

No matter how secure something is designed, physical access to the machine can break all protections. There are too many ways to compromise the bastion host. The easiest denial-of-service attack is to unplug the power. More advanced compromisations are changes in hardware, as have been performed in Microsoft^{TM}Xbox and in satellite receivers ([Ves02]).
Some of the most dangerous people to badly located bastion hosts are the ordinary people in the building, like cleaners, employees or students. A secure bastion host should be in a locked room, preferable in an alarmed room. Uninterruptible power is also to prefer if the service is critical. The environment in the room should offer air condition and ventilation.
Most of the network interfaces, like Ethernet and some token rings are designed to pick up all packets on the net, not only to packets addressed to the right interface. This is a great security hazard, due to snooping passwords and other secret information as in telnet and FTP sessions. If the bastion host is compromised, snooping has to be avoided. One solution to the problem is to put the bastion hosts on a perimeter network (DMZ), with no ordinary traffic. The traffic on the internal network is not seen by the perimeter network. Then if the bastion host is compromisedm, all it can see is traffic from and to itself. The solution is not totally secure, sensitive data might go out on the Internet to, but that is more of a security policy issue. If it possible each service or bastion host would be placed in a separate perimeter network. Another alternative is to put the bastion host on a perimeter network and have a virtual private network for each service ([Cha00]).

4.4 Operating Systems

As in all computer programs the *operating system* (OS) is the foundation for all applications. Without a stable and secure foundation the program will be unreliable. In programs where security comes in the first room this is even more important. The general principles when building a system for security reasons is to keep it simple and be prepared for attacks ([Cha00]). The more simple system, the easier it is to secure it. A bigger system is more complex and more complex systems are harder to secure ([Sch00]).

The second reason is the larger number of *lines of code* (LOC) that exist in a bigger system. More LOC gives more bugs, studies at Carnegie Mellon University have shown 5-10 defects/KLOC in normal code after integration and product testing ([Hum99]). It is important to note that threats to the system does not always come from the outside or maliciously contrived code, it might just as well come from bugs which may cause the rest of the system to behave incorrectly ([Kin01]).
The second principle, see section 4.4, was to be prepared for attacks. By assuming that someone will try to break the system and planning for the worst case scenario, it is more likely to avoid to be compromised.

When choosing OS the first thing to think of is what services the machine will be used for, the relevant tools (proxy servers, packet filtering systems etcetera) might not be available for all platforms. Today different Unix flavours are the most popular OS for Internet services, this because of high security and all applictions that are available. Another OS which can be used for bastion hosts is Windows NT. Windows NT is more complex and harder to secure than Unix and this is a big problem. But on the other hand if the user has more knowledge in Windows NT than Unix, Windows is a better choice ([Cha00]).

Would it be more secure to choose a proprietary OS than an open-source OS? Probably not or as stated in ([Cha00]) "paying money to a vendor does not guarantee that they care about security, only that they care about money". Different OS suppliers differ in the degree of openness about security and it is not always to prefer an OS that has good reputation about security holes. In the other end it is not always preferable to choose a system with many security holes. It is a better assumption that with a popular OS with many users, more security problems will be exposed and become public. But is this true for open-source OS? Open-source OS usually publish bugs and security holes on respectively web site and therefore, instead of choosing the OS with the least number of bugs, the OS with the more simple bugs/security holes would be preferable.

4.5 Keep Software Secure

When using software from other vendors, it is important to determine and evaluate updates, security fixes and patches to keep the system secure. Updates and other fixes exist due to the complexity of software. Many problems do not occur until the program is widespread. When a bug is discovered, many companies publish the bug on a web site or in mailing lists. The malicious people also look at these sites and lists to get a lead. There is often a gap between the release of a patch to the real update. This time gap is often exploited by malicious people. The challenge is to minimise the gap. The procedure to manage this is to be aware of the problem, to take immediate action and to permanently fix the problem when detected.

The first step is to gather and maintain a list of all resources where the bugs are listed. The most important information is often published on the vendors web sites. There are also mailing lists and USENET groups with important information. The problem with this type of information publishing is the volatility. The list must therefore be maintained and ensured to be up to date. To manage this a procedure and policies need preparation to monitor the information. Daily monitoring is recommended.

When an update is issued, it must be evaluated before it can be applied. The update might affect other parts of the system or the update itself might be vulnerable. When the update is performed, the system can be affected. There are programs to facilitate the evaluation phase. The programs show the difference in the system before and after the update, but a backup is always preferable before an update. After an update, documentation and cryptographic checksums on the software will identify unexpected changes ([CER01]).

Even if a good procedure for monitor updates are developed and the updates is evaluated according to the above statements, there are other aspects. When updating the software partially, the system gets fragmented and it is much harder to be in control of the system. In some cases it is better to change the whole system at once. This is of course not practical in all systems, but in smaller systems it is possible to do it in batch. Another aspect is the vendors trustworthiness. Always evaluate the update before a final

installation ([Ves02]).

4.6 Write Secure Code

Code is never secure, there are always bugs, but it can be more or less secure ([Hum99]). Code quality is something subjective, but there are processes, such as personal software process, to develop better code. Due to the proportionally of lines of code and the number of defects, less lines of code tend to produce less defects. Therefore perfection is not fulfilled when there is nothing more to add, but first when there is nothing to remove ([Ray01]). This do not imply that small and compact programs to provide secure code, never compromise security to get small sized code.

The most common security concerns come from buffer overruns [LeB00b]. This is not a big problem to overcome, but it requires careful developers. Buffer overruns is not only a common flaw, it is also one of the most dangerous, because it lets the intruder execute local command-line applications ([LeB00b]). If the code is written in a simple manner with many comments and spaces, it will be easier to review and debug. The commentation of the code is even more important when other people are involved in a project. Many problems occur when something unexpected happens. To prevent this, code written in a logical flow reduce the problem. Failures always occur, never underestimate that rule. When failure occur it is better to fail gracefully, than a total crash. Failure checks and assert functions are used to check all inputs to functions and expressions. A function is also much better designed if it is simple and short. Functions should not be too flexible, it is better with distinct functions. When code is extended too many times and the code has lost its original purpose, a phenomena called code rot occur ([LeB00a], [Sil01]).

If a system is critical, each part has to be tested and evaluated before it is inserted in the system. Then a higher degree of security is reached for the system, but the fusion of all parts can not be granted. To provide a secure system, the conglomerate has to be tested too ([Ves02]). The review of the code will get better if many people check the correctness of the code.

The review of the code will improve if as many people as possible read the code. Another advantage with letting people reviewing the code is that the programmers get more motivated to write good code if they know that more people will check the code.

4.7 Services

A secure bastion host needs to run some type of service to be usable. If firewalling and routing are excluded, the proxy services are the most used. Literally, a proxy is someone who is authorised to do something for another. In the computer world the proxy server acts the middle part. There are different types of proxies, caching and non-caching. Non-caching proxies are primarily used for security reasons, whereas the caching proxies are used for network performance.
A proxy should, if it possible, be designed to be transparent. The parts of a communication should not be conscious about the fact that they communicate with the proxy and not directly to each other. This adds security to the network, it is only the proxy that is exposed to the Internet, not the whole internal network. It also saves addresses in the address space ([Cha00]). All types of proxies can not be transparent, such as register proxies and similar. If a proxy is used it should not be any possibilities to bypass it, this might lead to great vulnerabilities.

A problem with some proxies, like Microsoft^{TM}Exchange SMTP proxy, is the exposure of a large interface to the outside. This makes it easier for intruders to make use of a compromisation. To prevent the possibility, a proxy for the proxy can be used, such as Qmail. Qmail only exposures a subset of the Microsoft^{TM}Exchange SMTP proxy ([Ves02]).

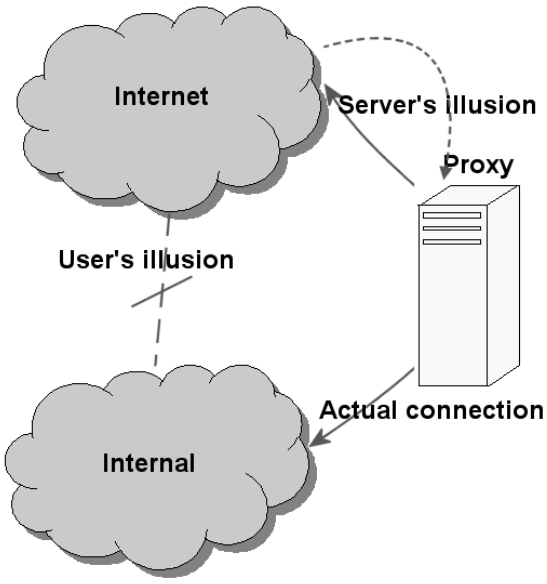

Figure 5: Redirection through a proxy server.

5 Specific Analysis

According to the topics studied in section 4 different strategies have been tested and evaluated. When practice the theories , many decisions must be taken to fulfill the criterias. The system design based on the characteristics from section 4.1 and more detailed studies have been performed. Economic- and open-source aspects are also discussed.

5.1 System Design

As discussed in section 4.1 it is a bigger challenge to build distributed systems than stand alone systems. In the case of the utility server it has to be distributed, due to the design of the firewall and to the specification. The other alternative to put the services in the firewall is not acceptable. The specific decisions in design is on the utility servers side.

Heterogeneity

The protocols for the utility server are well defined. It will use IP as network protocol and other protocols on top for management and the proxy services. The big decision is to choose operating system in such way that it is secure and maintainable and supports enough applications.

Openness

The open-source operating systems have a high degree of openness by nature. They are easy to extend and modify. The challenge is to implement openness in the design of the structure of the system. It has to be easy to add and change services. The boot procedure must also be modifiable.

Security

This is the main part of the projects system design. The system must be secure from bottom up to the top, no parts excluded. The decisions include everything from hardware, network location, operating system, design of the system to specific source code design.

Scalability

The total design of the utility server makes it very scalable. It is only to add more utility servers to make the system scale better. At a more detailed level there are quite many decisions to make. The hardware must be scalable, it might be necessary with more network interfaces or more

RAM. There must also be a uncomplicated way to upgrade and add more services. That part includes the design of the bootmenu and the structure of the internal operating system.

Failure handling

There are different types of failures that might occur in the utility server. The most fatal one is if it crashes. A crash is not just as simple as it seems to detect. It might be the network or something else on the way to the utility server. Some of the services are not critical and are only used for performance reasons. If a non critical service crashes, settings in the firewall can mask those failures. Those problems can be tolerated. The difficulty is to detect the failure. Other services are quite critical and must be detected and fixed if failures occur. More advanced techniques, such as redundant utility servers might also be used.

Concurrency

Most of the internal concurrency is handled by the operating system. The total design of the utility server with distributed nodes makes it concurrent by nature.

Transparency

There are two types of transparency challenges, transparency to the network and the proxies should be transparent to the users, if this is possible and required. Some proxies are not meant to be transparent. The network transparency is almost fulfilled with the original design of the system. The firewall only needs to know where the utility server is located. The other challenge is to make the proxies transparent to the users, with some exceptions. The proxies should not be visible to the users and the utility server solution must be designed to provide that. There are some network placement solutions and firewall extensions that make the utility server transparent to the users.

When the utility server is distributed with a firewall before and after, the packets gets filtered before they reach the utility server, see Figure 6. Then the traffic fulfills the least service characteristics. This must also be fulfilled in the configuration of the utility server.

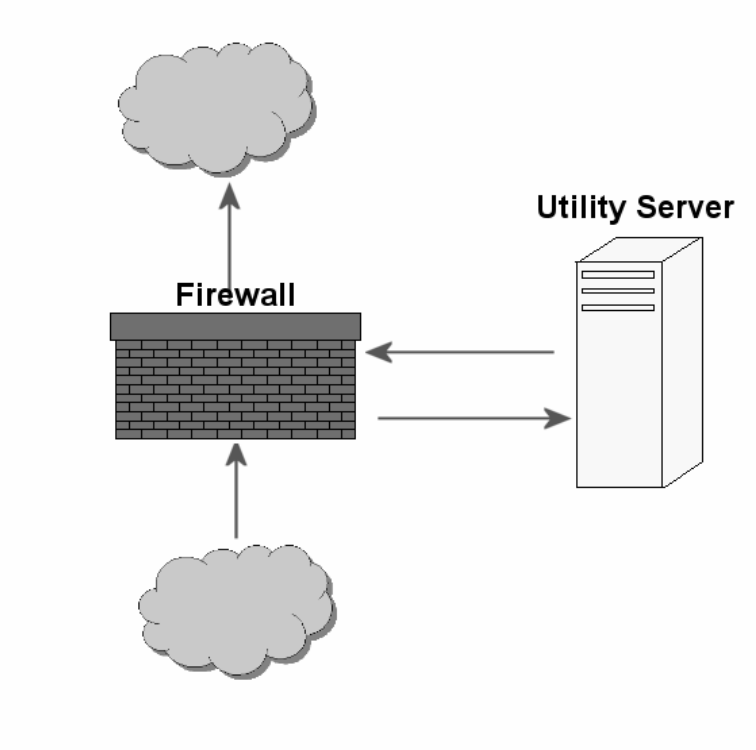

Figure 6: Least privilege for the traffic.

5.2 Hardware

The choice of hardware depends on the services that will run on the machine. The types of interfaces to choose are Ethernet based. If no bandwidth demanding services will run on the machine, it might be enough with a 10/100 Mbit interface. But in most cases a gigabit Ethernet interface is the most preferable. When caching proxies are used, multiple interfaces should be supported.

As discussed in section 4.2 it is better to avoid movable parts in the bastion host, but in some cases it is not possible. The hardened operating system can always be on a flash media or similar. In some cases with small and non-caching proxies it is possible to put all services and the operating system on the flash media. If a hard drive is needed, the implementation must detect failures on the hard drive. The size of the RAM is another aspect of the memory. In some cases it might be better to have a lot of RAM and use

it as a RAM drive, instead of a hard drive. The disadvantage is the loss of permanent memory. It is possible to use the flash media to permanent memory, but there is a upper bound of the number of insertions ([Apa02]). The speed of the CPU is usually not a bottleneck and as stated in section 4.2, a slower machine is not a tempting target to compromise. Of course, there are no reasons to choose such a slow CPU that it becomes the bottleneck. It should never reduce the throughput of the server.

5.3 Operating System

One goal with the project was to use a very hardened OS based on open-source and as mentioned before the OS had to be stable and secure. The fact that the OS and the programs running on top of the OS had to be open-source reduces the choice to Linux or open-source BSD distributions. There are many open-source distributions based on Linux (Debian, Mandrake, Red Hat, etc) or BSD (freeBSD, openBSD, netBSD) and the problem is to choose the OS that is the most secure and supports proxies to use. The utility server will be configured as an internal bastion host, but this do not decrease the needs of high security. Machines running as internal bastion hosts should be configured and protected more than normal internal hosts ([Cha00]).

5.3.1 *BSD

There are several different open-source distribution based on the BSD kernel. The most known is OpenBSD ([Ope02]), FreeBSD ([Fre03]) and NetBSD ([Net03]), where OpenBSD focus on security, FreeBSD on the free software and NetBSD on portability and networking.

picoBSD picoBSD is a distribution based on freeBSD. The goal with picoBSD is to fit a fully functional freeBSD system onto a 1.44 MB floppy disc ([Cai02]). There are pre-built floppys for bridge, net, dial, isp and router systems. To compile a one's own version a fully functional FreeBSD installation is needed. In this way the distribution can be adapted to the needs of the specific system.

compactBSD

CompactBSD ([Sim02]) is a distribution based upon openBSD. It is a tool

to build a snapshot of OpenBSD and burn it on a flash card. The tool is developed by FatPort ([Fat02]) and it is free to use under FatPointOEM licence, which is similar to the BSD licence. The tool is written in Python. CompactBSD is pretty easy to use, but it requires a lot of changes in the scripts to get it to work. When the script is modified, it downloads the necessary files with FTP from the OpenBSD download site. Then it creates a snapshot of the operating system and copies the files to a directory. In the last step the distribution is copied to the flash media.
The size of the compactBSD distribution is rather big compared to other alternatives. Another drawback is in the uncontrolled way it is installed and downloaded. Finally, the lack of good documentation is a problem.

Making a Distribution Manually

In all open-source operating systems it is possible to make your own distributions. Of course there are different ways for different operating systems and different methods. What is the opportunities to create a distribution manually? The greatest advantage is control, each file has to be reviewed before it is added. It is also possible to modify more than in an already created distribution.
In this section, how to reduce openBSD will be described. It is based on Manuel Kaspers guide how to reduce freeBSD ([Kas02]). First of all a normal installation of openBSD is necessary on a workstation or similar. The procedure of this can be found at ([Ope02]). It is not necessary to install packages such as xbase, game etcetera only a standard command-shell is needed. When the OS is installed the easiest thing to do is to create a directory where to put files needed for the reduced distribution e.g `/usr/miniOpenBSD`. In this directory a sub-OS will be created, so the first thing to do is to create directories.

```
.
|--bin
|--dev
|--etc
|--root
|--sbin
|--usr
```

```
|--bin
|--lib
|--libexec
|--sbin
```

When the directory tree is created the next thing to do is to copy the binaries over to `/usr/miniOpenBSD`. At [Kas02] is there a perl script `mkmini.pl` which takes a text file with the files to be copied to the destination directory e.g `/usr/miniOpenBSD`. Edit the text file to include the binaries wanted in the modified distribution and execute the perl script.
Before adding the kernel to the distribution it is a good idea to rebuild it to fit the needs. When the kernel is rebuild it can be compressed to save more space before it copied to `/usr/miniOpenBSD`.
Now the binaries need their depending libraries. For this purpose another perl script (`mklibs.pl`) at [Kas02] can be used. The script uses the command `ldd` (link dynamic dependencies) to find the libraries to the binaries. After running `mklibs.pl` a text file with necessary libraries is produced. This file can now be used with the first script `mkmini.pl` to copy the libraries to the right directory.
The easiest way to create the configuration files in `/miniOpenBSD/etc` is to select the one that is needed and copy them from `/etc`. Of course some of them has to be modified, for example `fstab` and `rc.conf`.
The next step is to create devices for the new distribution. one way to do is to copy the file `MAKEDEV` from `/dev` to `/usr/miniOpenBSD` and execute `sh MAKEDEV all` in the directory `/usr/miniOpenBSD`. If the distribution really has to be minimalistic it is possible to delete those devices not needed.
In this stage it is possible to copy the new distribution to the flash card, but that is not too good. The best thing is to create a image of the distribution and `dd` (convert and copy a file) it to the flash card. This way reduces the times the card is written to and saves the lifetime of the card. Another aspects to bear in mind is to keep log files on a hard drive or not at all. Swap-files on the flash card will also ruin the card pretty fast.

5.3.2 Linux

Linux is a clone of the Unix operating system written from scratch by Linus Torvalds ([ker03]). Linux is free and based on open-source with *general public license*(GPL) licence. There are a bunch of different distributions based on the Linux kernel, such as Mandrake, Debian, Slackware, Suse and Red Hat.

Debian with BusyBox and uClibc

Debian is a distribution based on the Linux core ([Deb03]). First of all the distribution must be installed on a machine ([Deb03]). When the distribution in installed it is a good idea to create a directory where the hardened distribution can be built. The next step is to compile a new kernel without unnecessary components and put it in the new directory. The second step is to copy the necessary binaries to the directory tree and the adherent libraries. Together with the devices and the `etc` directory a pretty functional operating system is built. Another way to go when building a distribution is to use specialised binaries and libraries for small or embedded systems.

BusyBox is one binary that replaces all common Unix utilities, like `cp` and `mv` ([And02]). The BusyBox is written with size optimisation and limited resources in mind. This makes the binary extremely small, only a few hundred kilobytes. It is also possible to exclude utilities at compile time, which makes it very modular. The drawback with BusyBox is the support of options to the utilities, but in the most cases the options are not needed. There are also possibilities to choose more options at compile time, but it makes it slightly bigger. BusyBox is used in many of the installer programs to Linux distributions, like Debian (boot floppies), Red Hat 7.2 and Slackware. BusyBox creates links and directorys when it is installed, which makes the installation procedure smooth and simple.

There are also modified libraries specialised for small systems. One example is uClibc, which is a very small GNU C library ([And03]). Most of the functions that is provided by glibc, are also implemented in uClibc. The advantage with uClibc is the size, it is much smaller than the common glibc and as stated in section 4.4 fewer lines of code less bugs. Most of the programs that works with glibc works fine with uLibc.
uClibc is much smaller because of the lack of support for other operating

systems, such as MS-DOS and Beos. It is only designed for Linux systems. Some of the least used functions provided by glibc, are not included in the library. Utilities like IPv6 and RPC support are also missing. This is often not a problem, until the special utility is needed. The biggest problem with uClibc is the complexity when installing and compiling new programs. All programs have to be recompiled with the uClibc-toolchain, which is pretty messy with larger programs, such as some proxy servers especially if the programs needs other libraries. Then the libraries also needs to be compiled with the uClibc library. Another small C library is dietLibC ([vL02]).

5.4 Services

The utility server will be designed to run many different services. The different types can be divided into caching proxies and non-caching proxies. A HTTP proxy is caching proxy, but an anti-virus proxy is not. The services have different demands on the server's hardware requirements. A caching proxy requires a lot of memory, whereas some of the other proxies only need to hold a state table in RAM. The reasons to run the services in an utility server are several. First of all it is not a good idea to have proxies in the critical part of the network protection, the firewall. Secondly the services are distributed the performance of firewall gets better.

Examples of different services in the utility server.

- HTTP proxy (URL filtering, Web cache).
- Mail relay (SMTP relay and filtering).
- Secure DNS service.
- SIP proxy.
- H.323 proxy.
- Anti virus proxy.

In which way the traffic is redirected from the firewall to the utility server differs depending on the type of proxy. If the utility server runs a HTTP proxy, the firewall only need to redirect the HTTP traffic to the utility server,

but with a SIP proxy it gets more complicated. When a SIP packet comes to the firewall, the packet must be examined and put into a state table. Then the packet is sent to the SIP proxy and the proxy makes some changes in the packet. Finally the packet is sent back to the firewall and it has to be examined again. This requires new implementations in the firewall kernel, which is not needed for example the HTTP proxy.

5.5 Handle the Market

When developing a new product to the market, there are several critical factors to be aware of before a final release. Without a thorough market research and strategical decisions threre are a high risk for design flaws. The first question is: Who will we sell the product to?
The answer to that question is where the company should press their marketing forces. Marketing can be made in different levels with the most general mass marketing to micro marketing where the products are specialised for each customer. In the between segment marketing and nisch marketing are found. The segment marketing is used on defined segments, while nisch marketing puts focus on smaller segments where the product can be successful.

The market can be divided in segments, like demographical, geographical, psychographical and behaviorgraphical segments. Of course there are other segments such as public sector and high-end companies, that are subsets of the previous. When the segments are defined and analysed according to competitors, potential competitors, substitute products and resellers, it is time to choose segment. One dimension is the attraction of the segment. It must be the right segment according to competitors and increasment in market positions. If the company does not have the power to win a segment, there is no use to establish [Won02]. Even if the company has the power to establish in the segment, it must be profitable.

To keep old customers and getting new, the company has to provide more value compared to its competitors. This can be done in two ways, lower prices or give the customers more advantages. Therefore it is important to do positioning of the product by differentiating. The problem is to differentiate the products in another way than the competitors. With big granularity the

different ways are:

- Product differentiating - The product is differentiated by performance, security, design, reliability and other characteristics.
- Service differentiating - Service can be differentiated by support, security, accessibility and quickness.
- Image differentiating - If the product does not distinguish between other products, a special brand can be a good differentiating.
- Personnel differentiating - This type of differentiation requires the company to do selective employments.

5.5.1 Open-Source

As all products, software has an utility value and a sales value. But the big difference from ordinary goods is the expectations on the software in the future. The customers buy the product not as it is, but because they are guaranteed that the product will be bug fixed and further developed. This is the state of art of software, no one trusts the software as it is. Patches will be released. Software products are therefore something in between of service oriented and manufacturing oriented.
When releasing a product as open-source the sales value is dramatically affected, but the utility value is constant. If the product is a secondary product or software for internal use there are significant arguments to make the source code available. To keep software for internal use up to date is both time and resource consuming, presumably totally meaningless. It is more effective to share the costs with other users. Another aspect is the share of risks. It is seldom the case that many employees have knowledge about the software. If someone quits the job, the knowledge about the program also might be gone. In normal manufacturing industry models to calculate the sales value are well defined. In manufacturing industry the addition method is the most used and in the service sector the division method is the most adopted one. For software it is more difficult to calculate the price according to already deployed methods and if the program is open-source, it is even more difficult.

The problems are not technical, rather social licence related. The licence models for open-source software often forbid distribution and changes in the code that only favour one part. If a company makes profit of selling modified open-source software, hackers might not like the project and therefore do not engage in the project. If the source code remains secret to the public it might have the effect of promoting competition. No one else can use the product and the sales value is protected. But there is no difference in functionality whether the source code is free or not.
Even if it is difficult to make profit on an open-source product, there are models for this. The main issue in many of these models is in which segment the company can earn the most money. For a more propitious segment position, the open-source software can be a loss leader or a market positioner. This model can be used when giving away free clients when the company selling servers.
When developing hardware, there is no reason to keep the driver secret to the public. In this case there is a great risk to keep the drivers secret, because of the enormous developing cost and the loss of market. With open-source drivers it is more likely that the driver is ported to other platforms. Another way to to make profit is to sell open-source distributions and offer support and a granted package of software. This model is used by companies like Red Hat™and Cygnus™.
The advantages with keeping the source code secret are well known. The company can sell the product alone on the market, but they have to develop everything by them self and the critical study over the code is less than in open-source products. This is the normal business model in the software market. But as mentioned before this model might not work, especially when the customers have expectation in future for the product or if the company wants to set a new standard. The reasons to make the code free can be characteristic by the following:

1. Reliability/stability/scalability are critical.
2. Correctness of design and implementation cannot readily be verified by means other than independent peer review.
3. The software is critical to the user's control of the business.

4. The software establishes or enables a common computing and communications infrastructure.

5. Key methods are well known by the engineers society ([Ray01]).

Seen from the customer's point of view, other factors are more important. If the company chooses conventional closed-source, they are in the hands of the supplier when it comes to bug fixes, updates, releases and improvements. The supplier knows this. The company has payed for the initial investment, not for the after sales. On the other hand, choosing open-source based software, the company itself can contribute in the development, if they find it necessary ([Ray01]).

6 Results and Implementation

The implementation is based on the earlier discussed theories and advice. Different choices have been made according to all parts of the system. Hardware, software and operating system are some of the parts where motivated choices had to be performed. The total system will be a critical part of an organisation's network services. Therefore the design of the system is very important and questions such as security and handling failovers must be handled.

6.1 System Design

The system design includes everything from network location to specific code design. When designing the system, the criterias listed in section 4.1, are recommended for a start. The total design must be permeated with that process of thought. First of all, a generic design of the system is done. In this case decisions are based on the specifications. In the original specification the utility server should be a stand alone server based on an open-source operating system, prepared to run different services. The mission was to test and evaluate different types of implementations and settings.

Heterogeneity

The networking is fulfilling the heterogeneity demands. All protocols used are based on well known RFCs, except the management protocol Netcon which is based on closed code, but it is possible to build own management tools based on other protocols. The operating system will be the Linux distribution Debian, discussed in section 6.4. The operating system is based on open-source and there are plenty of applications to Debian.

Openness

The design of the modified hardened operating system provides an easy way to maintain the system. Further an interface will be developed for remote upgrades. It is also easy to add new services, both to the system and to the bootmenu.

Security

The security of the kernel is just as high as it is for normal Debian installations. The surrounding system is very hardened and according to studies

discussed earlier, it should therefore be more secure. The total design such as hardware, network location etcetera is also designed with high security as a primary goal.

Scalability

The utility server will be scalable, because of the distributed design. It is possible to scale on service level, with different utility servers to each service or many utility servers to the same service. With different servers to each service there is no problem to redirect the traffic to the right service. But with two or more servers for the same service it will be a more complex design of the system. One solution might be a rotating DNS table in the local DNS server.

Failure handling

There is no specific failure detection in the utility server for the moment, excluding the failure checking in the source code. Better failure checking can be provided by status scripts that check whether the utility server or the services are running. The firewall can implement an automatic `ping` function to test if the network and/or the utility server is running. If a non-critical proxy fails, the traffic could be redirected and thereby masked.

Concurrency

When the utility server is distributed to one node, there is no more concurrency than the concurrency the operating system provides. But the design makes it possible to add more utility servers to provide concurrency on service level.

Transparency

The system itself is transparent to the user and it will be no difference with multiple utility servers. The network location also makes it transparent to the users. Transparency is also reached for the most of the services, but all services will not work with total transparency.

There are no users on the utility server and most of the common Unix utilities will be disabled, to provide least privilege. The server network location also guarantees the traffic and that only the right users can access the server. The specific decisions when it comes to network location, hardware etcetera are discussed below.

6.2 Network Location

The location on the network will be in a DMZ, see figure 7. A DMZ adds a additional layer of security to the utility server. If the server is compromised, all the intruders can see is packets to and from the utility server, nothing on the internal network. A higher degree of security and easier management will be reached if each service is on a separate VLAN.

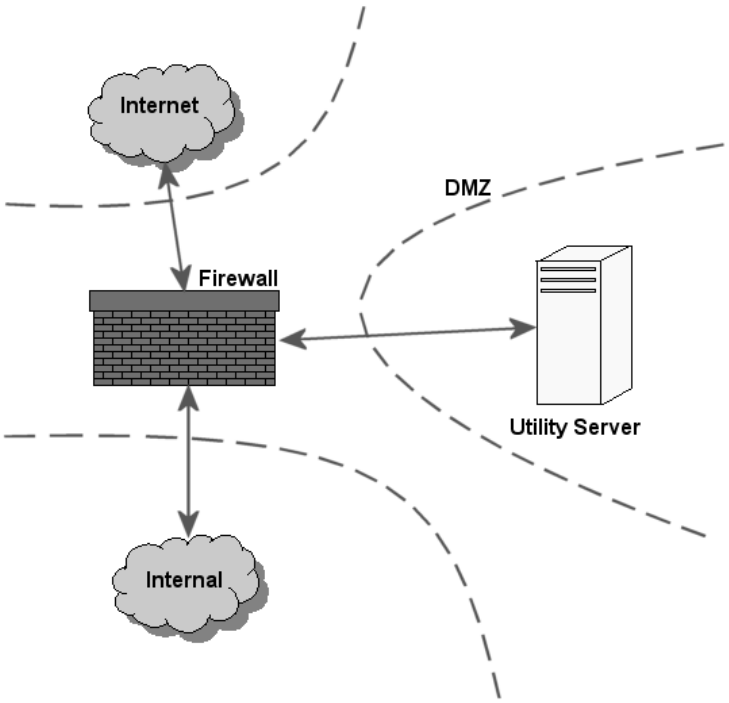

Figure 7: Network location of the utility server.

6.3 Hardware

The hardware requirements on the utility server vary a lot depending on the services, but it is not possible to have different configurations on different utility servers, due to the high development costs. The only difference should be between services running caching proxies and those running inspection prox-

ies, where the latter do not need any hard drives. The mixture of different services will determine the hardware requirements in the end. The network interface should be at least one Ethernet gigabit adapter, preferably Intel PRO/1000. This because of the better performance and the small difference in price.

The memory issue is harder, three decisions must be made. Type and size of flash media, type and size of possible hard drive and finally size of RAM. The flash media will be used to provide a secure operating system. It is worse if the operating system fails than if a service fails. The operating system and the surrounding environment require about 3,5 MB. The size of the services is harder to estimate. The smallest one, such as non-caching proxies, requires about 1-2 MB and a typical anti-virus proxy requires about 25 MB. If a hard drive is necessary, due to the services properties, a flash media size of 32-64 MB is enough, in other cases as a suggestion a bigger size is preferable. One problem with only using flash media is the logging. As mentioned in section 5.2 the flash media is sensitive to insertions, therefore a utility server must do the logging in a different way. For example it can be done over the Netcon protocol or syslog to an internal log server. In cases when caching proxies are used a SCSI hard drive is recommended. The size depends on the number of users in the system ([Pea02]). More users and higher bandwidth requires larger disk to cache all data. It is hard to estimate the needed size of a hard drive, about 20 GB would be enough for a 10 Mbit line ([Pea02]). More important is the disk random seek time and disk throughput.
The size of RAM is also depending on the services, but more than 512 MB is probably not necessary. To some services it would be possible with RAM disk instead of a hard drive. In those cases the size of the RAM must be adjusted to the needs of the services.
An alternative to hard drive is solid state flash drives, which use erasable NAND E2PROMs. There are several advantages with that solution. They are much more durable to vibrations and operating shocks and up to ten years data integrity is promised ([Mem02]). CPU power is in the most cases not the bottleneck. Somewhere between 1,2 GHz to 2,4 GHz Pentium, should be enough. Many services require less than that.

6.4 OS

The utility server will use a modified and very hardened operating system based on the Linux distribution Debian. Linux will be used instead of any *BSD distribution, because there are are more services available for Linux, than for BSD distributions. Another reason is the specialised tools and libraries available for Linux that are used to build small systems. There are no aspects of security when choosing Linux and Debian in favour to others, there are almost no differences between the open-source distributions. In the test implementations the hardened Debian distribution is quite smaller than a solution based on *BSD and this speaks for the Debian distribution. The smallest *BSD distributions, such as picoBSD and emBSD, are smaller than the hardened Debian version, but they are too complex and add too much of extra functions not needed by the utility server.

One goal with the utility server was to eliminate as many of the binaries as possible. With few binaries there are fewer possibilities for failures. In a normal distribution most of the binaries are the system commands, like `cp` and `mv`, so the first mission was to find alternatives and remove the unnecessary commands. One way is to just remove the corresponding binaries to the commands. That is possible but there will still remain many binaries. The other solution was the BusyBox project, which is described in section 5.3.2 ([And02]). BusyBox only contains one binary for all system commands. All commands are links to the BusyBox binary. This is a great advantage both to size and to maintenance. It is very easy to replace only one binary, instead of all system command binaries. The size is only about 200 KB, depending on how many commands the BusyBox should support. Another advantage is the possibility to exclude commands at compile time which eliminates many possible security holes. BusyBox is smaller, easier to maintenance, more secure and more flexible, therefore the BusyBox is the best solution for the utility server.
There are some other necessary binaries, like e2fsck, fsck and getty. They are not supported by BusyBox and must be added. Two binaries are also added to provide the boot menu. Each service also has it own binaries, but totally without services there are only six binaries in the base system.

A big problem with the minimal library uClibc is the lack of all functions and the need of recompiling all programs. If the system is built around uClibc and in the future a program needs a function that is not implemented in uClibc, the total system has to be rebuilt. That is not acceptable in commercial products. Situations where the vendor of a program do not want to compile the code with uClibc might also occur. Therefore is it better with a bigger library, such as glibc than a modified. The drawback will be the size and more lines of code, but on the other hand the code will be more tested.

6.5 Bootmenu

The system needs a bootmenu when it is shipped to customers. The bootmenu is written in C++ and is designed to look like the bootmenu on the Clavister firewalls. The graphics is from the ncurses library, with depending libraries. The menu is built from a text file, which describes the menu in a certain way. Other dynamic data are contained in the config file.

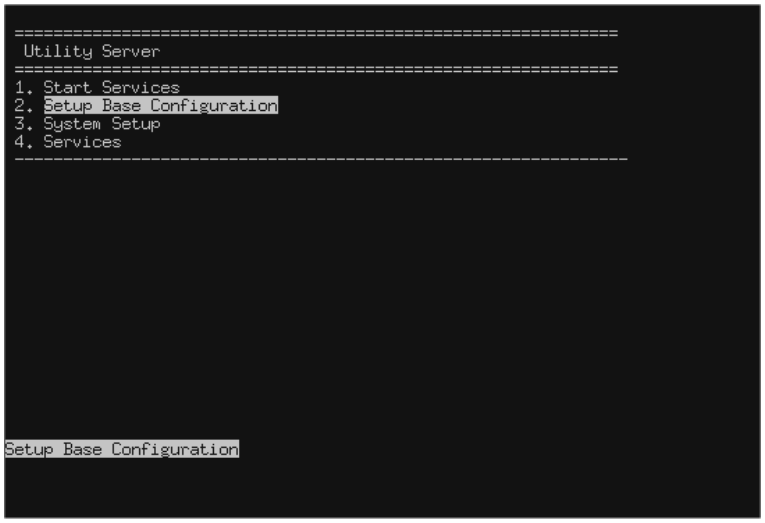

Figure 8: The first menu of the bootmenu.

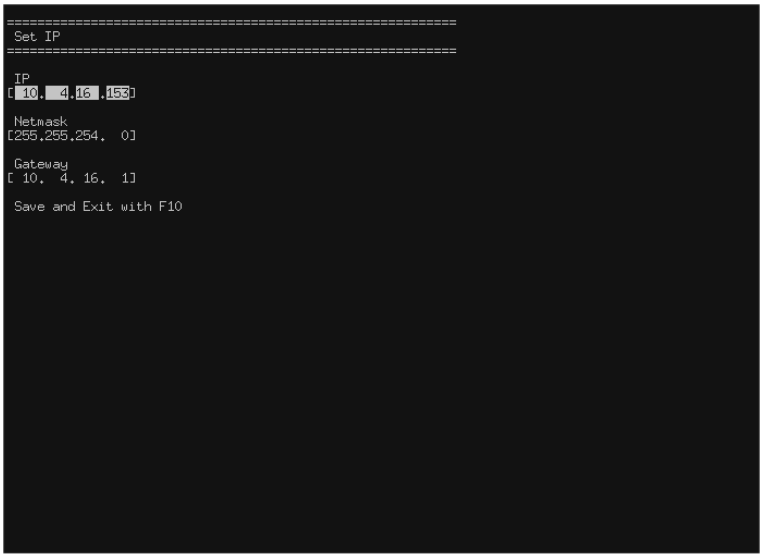

Figure 9: Base configuration menu.

The bootmenu is started with a config file as argument. The config file contains information about the services and network information. When saving configuration in the bootmenu, the config file is modified. The use of a config file makes the shipping and maintenance easier.

Example of config file to the bootmenu.

```
menufile=/etc/bootmenu/menu
services=3

[SERVICE1]
        name=SIP
        description=SIP proxy
        bin=/opt/partysip/bin/partysip -d 5
        start=yes

[SERVICE2]
        name=HTTP
        description=HTTP proxy
        bin=/opt/http/bin/http_proxy
        start=no

[SERVICE3]
        name=H323
        description=H323 proxy
        bin=/opt/h323/bin/h323_proxy
        start=yes

[NET]
        hostname=US1
        network=112.4.126.0
        broadcast=10.4.17.255
        ethdev=eth0
        ipaddr=112.4.126.153
        netmask=255.255.254.0
        gateway=112.4.126.1
```

The menu is built dynamicly from a text file, where the menu is described.
This makes the implementation more modular and adaptable.

Example of menu file to the bootmenu.

```
MENU "Utility Server"
  MENUITEM "Start Services" "Starting selected services" ""
    SUBMENU "Setup Base Configuration"
      MENUITEM "Set IP, Netmask and Gateway" "Network Configuration" "ip"
      MENUITEM "Return to Main Menu" "Back" "back"
    END
    SUBMENU "System Setup"
      MENUITEM "Set Time" "Set time" "time"
      MENUITEM "Set Date" "Set date" "date"
      MENUITEM "Reset to Factory Defaults" "Reset configuration" ""
      MENUITEM "Revert to Default Remote Managment Keys" "Use Default Keys" ""
      MENUITEM "Return to Main Menu" "Back" "back"
    END
    SUBMENU "Services"
      MENUITEM "Select services to use" "Select services" "services"
      MENUITEM "Configure Services" "Configure services" ""
      MENUITEM "Return to Main Menu" "Back" "back"
    END
END
```

6.6 Boot scripts

An essential part of the system is the boot procedure. To provide full control over the boot process, new boot scripts have been written. As always in secure systems the scripts must be written in a secure way as described in section 4.6. One goal was to minimise the procedure, mainly by security aspects but also concerning speed in case of an unexpected boot. The boot procedure is also dependent of the data in the config file. For that purpose the `bootmenu_config` file parse the config file and extracts the wanted data.
The etc directory including boot scripts.

```
.
|-- bootmenu
|   |-- bootmenu
|   |-- bootmenu_config
|   |-- config
|   `-- menu
|-- fstab
|-- init.d
|   |-- checkroot.sh
|   |-- defaultrcS
|   |-- rcS
|   `-- startservices.sh
|-- inittab
|-- ioctl.save
`-- mtab
```

6.7 Services

The number and types of services that should be used are very much depending on the demands of the market. The actual services are services that need to operate at application level. All services that needs to examine the packets at application level can be distributed to the utility server, with some exceptions. This will add extra security and performance improvements. The security improvements comes from the fact that some of the protocols are still drafts and immature, but needed by organisations and therefore used. When moving the service from the critical firewall to a less critical utility

server the security increases. The performance is increasing when the firewall do not need to examine packets at application level and can concentrate on the primary goal.

Some services are much more easier to implement in the system than others. A service that use a protocol which only needs redirection in the firewall, such as HTTP proxy is an example of an easy service to implement in the system. Services that need modifications in the firewall, such as SIP and H.323 will take longer time to offer the market.

On the utility servers side there is no problem to install new services. The server is built in a scalable way to make it easy for upgrading and adding new features. But before a new service is added, it has to be evaluated and tested. The hardware requirements are also an aspect to reconsider before an eventual installation.

The suggestion is to evaluate and test as many services as possible and install the ones that fulfill the security requirements and the markets demands. A fast way to minimise time to market is to install the services that do not need any changes in the firewall kernel.

6.8 The Market

The greatest advantage for Clavister with a utility server is to provide a complete product line solution. From offering security products like firewalls and VPNs, other service oriented boundary products can be incorporated in the product line. It is easier to sell a complete solution to the customers, than parts of a solution, especially when all products can be managed and administrated from the same program.

Segmentation of the utility server may be two sided. It is recommended to either find a segment for the utility server or use the utility server to tight up the segment. The first case is complicated. Is it possible to find a segment with high attraction for only this product? Probably not. It is not possible to do an effective segmentation with realistic and substantial values. The suggestion is to use old segments and use the new product as a way to fortify positions in the existing segments.

To offer a complete product line, Clavister can differentiate all of their prod-

ucts in a complete solution. In that way Clavister will offer a higher degree of service value to the customers. It is not a good idea to market the utility server as a stand alone product. Another way to provide more customer value is free support. Support on the utility server will be more resource demanding than support on the ordinary product, because of the different services that will be used on the server. One solution is to only provide support on configuring the utility server's setup procedure together with Clavister firewalls and not on the specific services on the server. The problem with that solution is it that might appear as if Clavister do not know and stand for its product. Then it might be better to chose a set of services where Clavister provides full support.

If software is categorised as a service sector, free support is suicide for a company. The only income to the company will be the sales value. When offering free support, the service must be included in the sales price. If the service is not included in the price, the product can be used as a loss leader and a way to differentiate the product. Satisfied customers may also generate income in the future.

The utility server is partial based on open-source and to keep it up to date and maintain the parts of the server, decisions must be made if all of the code should be open-source. As discussed in section 5.5.1 there are different ways to make profit on open-source products. The utility server will be more of a loss leader, than a key product and according to [Ray01] that is a golden opportunity to make the source free. This might be the case, but there are other circumstances that influence the decision.

If the characteristics for letting the code go free are studied it is easier to make a decision.

Reliability/stability/scalability are critical.

This criteria it totally fulfilled for the utility server, especially reliability and stability.

Correctness of design and implementation cannot readily be verified by means other than independent peer review.

It is possible to do the independent review in house and the correctness would not increase if the source was free.

The software is critical to the user's control of the business.

The customer does not need this product to make business, but it helps and together with a firewall solution it tends to be critical. Alone the utility server is not critical for secure business.

The software establishes or enables a common computing and communications infrastructure.

This implies a customer's perspective. The customers do not want to be tied up to one supplier. The utility server is not such a product, there are other ways for a company to provide the same services as in the utility server. The difference is the total solution.

Key methods are well known by the engineers society.

This is true for the utility server. The strength of the utility server is not in the code. There are no secrets in the code or the design. It is only based on scientific research.

Only two characteristics are fulfilled to make it open-source, the first and the last. The first is very important because the product will be a part of a security solution, but it can not by itself have such a great affect on the decision. The last point, if the key methods are well known by the engineers, is also important but not in the same way. The methods are well known to the public, but someone has to implement it. That has already been done and it is a great advantage over the competitors.

Problems that may occur if the source code is free are:

- Lower sales price.
- Loosing control of the project.
- Loosing market share on the utility server market.
- Other companies copy and sell the solution.

It is a big step to manufacture the product as open-source and there are plenty of advantages and disadvantages. For the moment, future development should be based on a secret code base. If the number of services increases cumbersome, other decisions can be taken. In this case when the source is free, it is extremly important with very good administratively tools in the manager.

7 Future work

The utility server is not yet produceable, there are parts of the system that must be implemented to provide a fully functional system. The most important parts are listed below.

- Better inheritance in bootmenu. - Reconsider to change the code design of the bootmenu.
- More failure checking. - More and better failure checking, both in the code and in more generic terms of the utility server.
- Mount system read-only after booting and configuring. - This enhance the lifetime of the flash media, but the system has to be read-write when configurating.
- Make the flash bootable. - Write new LILO.
- Make the system reproduceable for appliance models. - A procedure must be developed for easy installation.
- Implement Netcon and Netcon agent in the utility server. - Will be used for upgrading and logging of the system.
- Write a new license according to the other licences. - There is a mixture of licence models with different OS and services.
- Implement Syslog logging. - The system should be able to use Syslog or Netcon to do logging.
- Implement VLAN feature to each service on the utility server. - Makes it easier to use the router and filtering possibilities in the firewall.
- Enhance the bootmenu. - Implement all alternatives.
- Implement logging functions.
- Write status script. - Check if services are up and other status. Should be self repaired.

- Add features to search on names in config file. - Will make the design better of the config file.

8 Summary and Conclusion

In this thesis a description of design requirements for secure bastion hosts and a test implementation based on an open-source operating system has been evaluated. The test implementation is based on security theories and recommendations.

Security theories, like encryption theory, are used in techniques such as SSL. The theories are often verified and secure, but bad implementations might harm the security. The techniques are used as parts in systems and to provide high security, the system itself must be secure.

There are mainly three different types of firewalls, which operates at different levels in the protocol stack. With a lower level the firewall reach high performance, but it gets static. The opposite is valid for firewalls operates at higher levels. The firewalls are manufactured in different ways like software, appliance or hardware firewalls. When more services are implemented in the firewalls, the system can be designed in different ways with the services incorparated in the firewall or distributed to external servers.

When designing secure bastion hosts, all parts of the system have to be secure. This includes physical location, network location, hardware, operating system, applications, secure source code and secure system design. It is a big difference between security theories and building secure systems.

The hardware choices will be based on the general theories for secure systems. Many different operating system have been tested and evaluated. The best choice was Linux Debian distribution because it fulfilled the system design goals. The only interface for the user will be the bootmenu. The bootmenu will therefore be dynamic and flexible.

Instead of only selling firewalls and VPN products, it is possible to manufacture a total product line for firewall solutions. This makes the segment tighter and the company will be more attractive because of the increased customer value. The criterias to make the product market as open-source are not fulfilled for the moment.

With the utility server the load of the firewall will be reduced and the flexibility will increase because of the ease to implement new services in the utility server. The system will also get a higher degree of security because of the

distributed solution.

References

[And02] E. Andersen. *BusyBox - The Swiss Army Knife of Embedded Linux.* Webpage, 26 October 2002, <http://www.busybox.net/>, 2003-01-10.

[And03] E. Anderssen. *uClibc - a C library for embedded systems.* Webpage, 10 January 2003, <http://www.uclibc.org/>, 2003-01-14.

[Apa02] Apacer. *Apacer CompactFlash™ Memory Card.* Webpage, 14 October 2002, <http://193.13.79.5/pdf-filer/-CF-128MB-VER.pdf>, 2003-01-09.

[Bla00] B. Blakley. *The Three Myths of Firewalls*, 18 January 2000, <http://www.mit.edu/kerberos/www/firewalls> 2002-12-12.

[Cai02] N. Caille. *PicoBSD*, 14 July 2002, <http://www.picobsd.org/>, 2003-01-14.

[CER01] CERT Coordination Center. *Keep operating systems and applications software up to date.* Webpage, 30 April 2001, <http://www.cert.org/security-improvement/practices/-p067.html>, 2003-01-08.

[Cha00] E. D. Zwicky S. Cooper D. B. Chapman. *Building Internet Firewalls, second edition.* O'Reilly, 2000.

[Cla01] Clavister AB. *User's Guide, Firewall 7.* Clavister AB, 2001.

[CSR02] CSRC Computer Security Resource Center at National Institute for Standards and Technology. *Advanced-Encryption Standard.* Webpage, 2002, <http://csrc.nist.gov/-encryption/aes/>, 2002-12-18.

[Deb03] Debian. *Debian.* Webpage, 14 January 2003, <http://www.debian.org/>, 2003-01-14.

[Dew96] P. DEWAN. *Bell LaPadula Model.* Webpage, 4 November 1996, <http://www.cs.unc.edu/ dewan/242/f96/notes/-prot/node13.html>, 2002-12-07.

[Ell87] J. H. ELLIS. *The history of Non-Secret Encryption.* Webpage, 1987, <http://www.cesg.gov.uk/publications/media/-nsecret/ellis.pdf>, 2002-12-07.

[Fat02] FATPORT. *FatPort*, 7 October 2002, <http://www.fatport.com>,2002-12-10.

[Fre03] FREEBSD. *FreeBSD*, 01 January 2003, <http://www.freebsd.org/>, 2003-01-14.

[HKU97] HKUST CYBERSPACE CENTER THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY. *Guide to Firewall Configuration.* Webpage, 1 November 1997, <http://www.cyber.ust.hk/-handbook4/01_hb4.html>, 2003-01-03.

[Hum99] W. S. HUMPHREY. *Bugs or Defects?* Webpage, 1999, <http://interactive.sei.cmu.edu/news@sei/-columns/watts_new/1999/March/watts-mar99.pdf>,-2003-01-08.

[itw02] ITWORLD.COM. *What Kind of Firewall Do You Need?* Webpage, 4 April 2002, <http://www.itworld.com/nl/unix_sec/-04042002/pf_index.html>, 2003-01-02.

[Kas02] M. KASPER. *miniBSD - reducing FreeBSD*, 4 October 2002, http://www.neon1.net.

[ker03] KERNEL.ORG. *The Linux Kernel Archives.* Webpage, 14 January 2003, <http://www.kernel.org/>, 2003-01-14.

[Kin01] G. COULOURIS J. DOLLIMORE T. KINDBERG. *Distributed Systems, Concepts and Design, Third Edition.* Addison Wesley, 2001.

[LeB00a] D. LeBlanc. *Good Programming and the Rules for Writing Secure Code.* Webpage, 27 2000, <http://www.ntsecurity.net/-Articles/Index.cfm?ArticleID=9201>, 2003-01-09.

[LeB00b] D. LeBlanc. *Secure Services.* Webpage, 27 June 2000, <http://www.ntsecurity.net/Articles/-Index.cfm?ArticleID=9195>, 2003-01-09.

[Mem02] Memtech. *Memtech.* Webpage, 2002, <http://www.memtech.com/>, 2003-01-10.

[Net01] Netscreen. *Stateful-Inspection Firewalls: The NetScreen Way An overview of firewall technology and how NetScreen implements.* Webpage, 2001, <http://www.netscreen.com/>, 2003-01-07.

[Net03] NetBSD. *NetBSD Project*, 12 January 2003, <http://www.netbsd.org/>, 2003-01-14.

[Ope02] OpenBSD. *OpenBSD - Free, Functional and Secure*, 10 December 2002, <http://www.openbsd.org/>, 2003-01-05.

[Pea02] O. Pearson. *Squid - A User's Guide.* Webpage, 2002, <http://squid-docs.sourceforge.net/latest/html/>,-2003-01-09.

[Ray01] E. S. Raymond. *The Cathedral and the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly, 2001.

[Riv92] R. Rivest. *Request for Comments: 1321,The MD5-Message-Digest Algorithm.* Webpage, April 1992, <http://www.ietf.org/rfc/rfc1321.txt>, 2002-12-07.

[Ros01] J. F. Kurose K. W. Ross. *Computer Networking. A Top-Down Approach Featuring the Internet.* Addison Wesley, 2001.

[SAN01] SANS Institute Resources. *The SANS Security Policy-Project.* Webpage, 2001, <http://www.sans.org/newlook/-resources/policies/policies.htm>, 2002-12-09.

[Sch96] B. SCHNEIER. *Applied Cryptography.* John Wiley & Sons, Inc., 1996.

[Sch00] B. SCHNEIER. *Secrets and Lies.* John Wiley & Sons, Inc., 2000.

[Sil01] SILVERLINE TECHNOLOGIES. *Basic Principles for Security Design.* Webpage, 01 January 2001, <http://www.silverline.com/-techFusion/src/pdf01_page_02.html>, 2003-01-09.

[Sim02] K. SIMPSON. *CompactBSD*, 20 August 2002, <http://sourceforge.net/projects/compactbsd>, 2002-12-10.

[Sur01] J. SURYANARAYANA. *Digital Signatures for SOAP messages.* Webpage, 15 March 2001, <http://www6.software.ibm.com/-developerworks/education/ws-dsst/index.html>, 2002-04-10.

[Tre02] W. TREESE. *Security Implications of Web Services.* Webpage, 04 April 2002, <http://www.ietf.org/html.charters/-tls-charter.html>, 2002-03-07.

[Ves02] J. VESTBERG. *Interview with John Vestberg, Clavister, Vice President Technology.* Interview, 16 December 2002.

[vL02] F. VON LEITNER. *diet libc - a libc optimized for small size.* Webpage, 14 October 2002, <http://www.fefe.de/dietlibc/>, 2003-01-14.

[web02] WEBOPEDIA.COM. *Server Applicance.* Webpage, 8 May 2002, <http://www.webopedia.com/>, 2003-01-06.

[Won02] P. KOTLER G. ARMSTRONG J. SAUNDERS V. WONG. *Principles of Marketing.* Prentice Hall, 2002.

A Example of Security Policy

This example of a security policy is taken from [SAN01].

InfoSec Acceptable Use Policy

1.0 Overview

InfoSec's intentions for publishing an Acceptable Use Policy are not to impose restrictions that are contrary to <Company Name>. established culture of openness, trust and integrity. InfoSec is committed to protecting <Company Name>'s employees, partners and the company from illegal or damaging actions by individuals, either knowingly or unknowingly. Internet/Intranet/Extranet-related systems, including but not limited to computer equipment, software, operating systems, storage media, network accounts providing electronic mail, WWW browsing, and FTP, are the property of <Company Name>. These systems are to be used for business purposes in serving the interests of the company, and of our clients and customers in the course of normal operations. Please review Human Resources policies for further details. Effective security is a team effort involving the participation and support of every <Company Name> employee and affiliate who deals with information and/or information systems. It is the responsibility of every computer user to know these guidelines, and to conduct their activities accordingly.

2.0 Purpose

The purpose of this policy is to outline the acceptable use of computer equipment at <Company Name>. These rules are in place to protect the employee and <Company Name>. Inappropriate use exposes <Company Name> to risks including virus attacks, compromise of network systems and services, and legal issues.

3.0 Scope

This policy applies to employees, contractors, consultants, temporaries, and other workers at <Company Name>, including all personnel affiliated with third parties. This policy applies to all equipment that is owned or leased by <Company Name>.

4.0 Policy

4.1 General Use and Ownership 1. While <Company Name>'s network administration desires to provide a reasonable level of privacy, users should be aware that the data they create on the corporate systems remains the property of <Company Name>. Because of the need to protect <Company Name>'s network, management cannot guarantee the confidentiality of information stored on any network device belonging to <Company Name>.

2. Employees are responsible for exercising good judgement regarding the reasonableness of personal use. Individual departments are responsible for creating guidelines concerning personal use of Internet/Intranet/Extranet systems. In the absence of such policies, employees should be guided by departmental policies on personal use, and if there is any uncertainty, employees should consult their supervisor or manager.

3. InfoSec recommends that any information that users consider sensitive or vulnerable be encrypted. For guidelines on information classification, see InfoSec's Information Sensitivity Policy. For guidelines on encrypting email and documents, go to InfoSec's Awareness Initiative.

4. For security and network maintenance purposes, authorised individuals within <Company Name> may monitor equipment, systems and network traffic at any time, per InfoSec's Audit Policy.

5. <Company Name> reserves the right to audit networks and systems on a periodic basis to ensure compliance with this policy.

4.2 Security and Proprietary Information

1. The user interface for information contained on Internet/Intranet/Extranet-related systems should be classified as either confidential or not confidential, as defined by corporate confidentiality guidelines, details of which can be found in Human Resources policies. Examples of confidential information include but are not limited to: company private, corporate strategies, competitor sensitive, trade secrets, specifications, customer lists, and research data. Employees should take all necessary steps to prevent unauthorised access to this information.

2. Keep passwords secure and do not share accounts. Authorised users are responsible for the security of their passwords and accounts. System level passwords should be changed quarterly, user level passwords should be

changed every six months.
3. All PCs, laptops and workstations should be secured with a password-protected screen saver with the automatic activation feature set at 10 minutes or less, or by logging-off (control-alt-delete for Win2K users) when the host will be unattended.
4. Use encryption of information in compliance with InfoSec's Acceptable Encryption Use policy. 5. Because information contained on portable computers is especially vulnerable, special care should be exercised. Protect laptops in accordance with the .Laptop Security Tips.. 6. Postings by employees from a <Company Name> email address to newsgroups should contain a disclaimer stating that the opinions expressed are strictly their own and not necessarily those of <Company Name>, unless posting is in the course of business duties.
7. All hosts used by the employee that are connected to the <Company Name> Internet/Intranet/Extranet, whether owned by the employee or <Company Name>, shall be continually executing approved virus-scanning software with a current virus database. Unless overridden by departmental or group policy.
8. Employees must use extreme caution when opening e-mail attachments received from unknown senders, which may contain viruses, e-mail bombs, or Trojan horse code.
4.3. Unacceptable Use The following activities are, in general, prohibited. Employees may be exempted from these restrictions during the course of their legitimate job responsibilities (e.g., systems administration staff may have a need to disable the network access of a host if that host is disrupting production services). Under no circumstances is an employee of <Company Name> authorised to engage in any activity that is illegal under local, state, federal or international law while utilising <Company Name>-owned resources. The lists below are by no means exhaustive, but attempt to provide a framework for activities which fall into the category of unacceptable use. System and Network Activities The following activities are strictly prohibited, with no exceptions: 1. Violations of the rights of any person or company protected by copyright, trade secret, patent or other intellectual property, or similar laws or regulations, including, but not

limited to, the installation or distribution of "pirated" or other software products that are not appropriately licensed for use by <Company Name>.
2. Unauthorised copying of copyrighted material including, but not limited to, digitisation and distribution of photographs from magazines, books or other copyrighted sources, copyrighted music, and the installation of any copyrighted software for which <Company Name> or the end user does not have an active license is strictly prohibited.
3. Exporting software, technical information, encryption software or technology, in violation of international or regional export control laws, is illegal. The appropriate management should be consulted prior to export of any material that is in question.
4. Introduction of malicious programs into the network or server (e.g., viruses, worms, Trojan horses, e-mail bombs, etc.).
5. Revealing your account password to others or allowing use of your account by others. This includes family and other household members when work is being done at home.
6. Using a <Company Name> computing asset to actively engage in procuring or transmitting material that is in violation of sexual harassment or hostile workplace laws in the user's local jurisdiction.
7. Making fraudulent offers of products, items, or services originating from any <Company Name> account.
8. Making statements about warranty, expressly or implied, unless it is a part of normal job duties.
9. Effecting security breaches or disruptions of network communication. Security breaches include, but are not limited to, accessing data of which the employee is not an intended recipient or logging into a server or account that the employee is not expressly authorised to access, unless these duties are within the scope of regular duties. For purposes of this section, "disruption" includes, but is not limited to, network sniffing, pinged floods, packet spoofing, denial of service, and forged routing information for malicious purposes.
10. Port scanning or security scanning is expressly prohibited unless prior notification to InfoSec is made.
11. Executing any form of network monitoring which will intercept data

not intended for the employee's host, unless this activity is a part of the employee's normal job/duty.
12. Circumventing user authentication or security of any host, network or account.
13. Interfering with or denying service to any user other than the employee's host (for example, denial of service attack).
14. Using any program/script/command, or sending messages of any kind, with the intent to interfere with, or disable, a user's terminal session, via any means, locally or via the Internet/Intranet/Extranet.
15. Providing information about, or lists of, <Company Name> employees to parties outside <Company Name>.
Email and Communications Activities 1. Sending unsolicited email messages, including the sending of "junk mail" or other advertising material to individuals who did not specifically request such material (email spam).
2. Any form of harassment via email, telephone or paging, whether through language, frequency, or size of messages.
3. Unauthorised use, or forging, of email header information.
4. Solicitation of email for any other email address, other than that of the poster's account, with the intent to harass or to collect replies.
5. Creating or forwarding "chain letters", "Ponzi" or other "pyramid" schemes of any type.
6. Use of unsolicited email originating from within <Company Name>'s networks of other Internet/Intranet/Extranet service providers on behalf of, or to advertise, any service hosted by <Company Name> or connected via <Company Name>'s network.
7. Posting the same or similar non-business-related messages to large numbers of Usenet newsgroups (newsgroup spam).

5.0 Enforcement

Any employee found to have violated this policy may be subject to disciplinary action, up to and including termination of employment.

6.0 Definitions

Term Definition
Spam Unauthorised and/or unsolicited electronic mass mailings.

7.0 Revision History