UNIVERSITY OF OSLO
Department of Informatics

Adaptive distributed firewall using intrusion detection

Lars Strand
UniK University Graduate Center
University of Oslo

lars (at) unik no

1. November 2004

ABSTRACT

Conventional firewalls rely on a strict outside/inside topology where the gateway(s) enforce some sort of traffic filtering. Some claims that with the evolving connectivity of the Internet, the traditional firewall has been obsolete. High speed links, dynamic topology, end-to-end encryption, threat from internal users are all issues that must be addressed. Steven M. Bellovin was the first to propose a "distributed firewall" that addresses these shortcomings.

In this master thesis, the design and implementation of a "distributed firewall" with an intrusion detection mechanism is presented using Python and a scriptable firewall (IPTables, IPFW, netsh).

PREFACE

This thesis is written as a part of my master degree in Computer Science at the University of Oslo, Department of Informatics. The thesis is written at the Norwegian Defence Research Establishment (FFI).

Scripting has been one of my favourite activities since I first learned it. Combined with the art of Computer Security, which I find fascinating and non-exhaustive, it had to be an explosive combination. My problem next was to find someone to supervise me.

This is where Professor Hans Petter Langtangen at Simula Research Laboratory and Geir Hallingstad, researcher at FFI, stepped in. Hans Petter Langtangen is a masterful scripting guru and truly deserves the title "Hacker". Geir Hallingstad is expert in the field of computer/network security and gave valuable input and support when designing this prototype. Huge thanks to them both.

I would also like to thank Ronny Windvik, researcher at FFI, for *lots* constructive help during the last intensive months of writing.

Also a special thanks to Camilla Rakvåg, for her patience and support during my many late hour hacking session.

Lars Strand,
November 1st, 2004 - Kjeller

Contents

List of Figures

Chapter 1

Introduction

"Life was simple before World War II. After that, we had systems."

— *Rear Admiral Grace Hopper*

1.1 Background

Computer/Network Security has not always been a hot topic. In the early days of computers, there was no need for protecting computers from malicious crackers; *"The entire art of hacking relied on intellectual openness and trust"* states president of FSF, Richard Stallman in [69]. When ARPANET arrived, the predecessor of today's Internet, it was designed to be reliable, not secure. It was a research and academic network focusing on openness and exchange of information.

After the arrival of Internet, there has been an explosive growth in the number of connected hosts. With the coming of e-commerce, commercial firms are relying more and more on Internet. Even large institutions which are critical to society are merge to the Internet; The U.S. Federal Reserve[1], which handles all the banks money transfers within the U.S., proclaimed on the 15th of August 2004 that it will no longer be using the closed banking network to do its transactions. Instead it will use the Internet.

Not only large firms/institutions become connected, also an increase in "gadgets"/"devices" that are online. Most of the new cellular phones can be used to check email and surf on the Internet. In the future, a person may log into a home server that controls the heat, light and may even look and see what's in the refrigerator. If dinner is wanted later that evening, an order to the local store may be placed. When the groceries are delivered to the door, an order of 50 bottles of beer is delivered instead, since the refrigerator had been cracked and a bogus order placed instead of the real one.

At the same time, the FBI says Cyber Crime is the fastest growing areas. This forces a strong focus on security; as the Internet changes, so does the focus on security. Computer security is no longer a luxury reserved the military and the government. With the Internet becoming critical to economy and an increased connectivity of institution providing sensitively information, like medical information, systems must certify to an acceptable level of security. *"Systems having no security are unacceptable in most environments today"* says Bishop in [8].

[1] U.S. Federal Reserve homepage: `http://www.federalreserve.gov/`

1.2 Problem

The idea for this master thesis came after I took a scripting course held by Professor Hans Petter Langtangen. I felt there was an important tool missing in the open source community; a decent distributed firewall management tool for the GNU/Linux operating system. I started sketching out a basic design and made a feature list. Even though some of the details have changed since the initial sketch, the main idea remains the same.

After I did some searching on the Internet, I was surprised when I found that Steven Bellovin already had outlined (more clearly) my first thoughts on this issue in his paper "Distributed Firewall" [5].

The traditional firewall, as it is deployed and used today, suffers from several shortcomings. The biggest drawback is perhaps the lack of protecting the *inside* traffic: A malicious user may launch an attack on the companies internal servers *from the inside*, since a traditional firewall only protect the traffic going though the Internet gateway. *A firewall can't filter traffic it doesn't see.*

The second limitation the traditional firewall suffers from is the evolving of more *mobile users*. An employee is no longer restricted to the company's office well protected by the companies firewall. He may be out talking to the customers, at home or on travel — the employee is *mobile*. At the same time, he may acquire companies resources like mail and accessing files/databases. *A traditional firewall can't protect a mobile user.*

1.3 Purpose

The purpose of the master thesis is to design and implement a secure distributed firewall tool for GNU/Linux. By using IPTables as policy language (firewall) and intrusion detection to detect compromised hosts.

The firewall tool consists of two main parts: the management node (master) and the client. The master is running on the management node and has the ability to connect to and give commands to all clients. The clients enforce any command given from the management node and send feedback back to the management node.

The commands given from the management node are firewall rules (using IPTables). Since these firewall rules are distributed as shell-script, any operating system with support for a scriptable firewall may be used.

By deploying and using the firewall tool, a system administrator in charge of small to medium sized networks, may manage the clients firewall and intrusion detection more easily. But the main purpose of this firewall tool is to meet the shortcomings of the traditional firewall. This includes *minimizing the threat from internal users*, by protecting internal hosts as well as the gateway. It also includes protecting mobile hosts, not located inside the corporate network.

1.4 Limits

The original feature list took longer to implement than first thought. Some design issues were proving difficult to implement given the time available. This includes the *adaptive* firewall based on *network* intrusion detection: That the intrusion detection manipulates the firewall rules directly without the interaction of the system administrator. The design for how this can be implemented is explained. The firewall in this thesis is still adaptive, but rules are given manually.

A graphical interface is not implemented; instead the system administrator uses a command line based shell to administer the firewalls. However the implementation is based on a modular design, so developing a graphical interface should be pretty straightforward.

This master thesis has not only been an exercise in programming; it has also been a guided tour in the art of computer- and network security, exploring the many pitfalls to avoid when trying to desig a secure application.

1.5 Chapter overview

Chapter 2 gives some background information on computer/network security in general and tries to explain what computer security *is* and what building blocks it consist of. Since computer security is response to certain security threats, the most common attacks are explained. The firewall is the first line of defence against these security attacks. The most essential security mechanisms used to design this firewall tool are also explained.

In chapter 3, firewall technology is explained. Different types of firewalls, problems concerning the traditional firewall, and what a "distributed firewall" is and what problems it solves are discussed.

Intrusion detection is briefly covered in Chapter 4. Various intrusion detection mechanism are explained and problems are discussed. Does a "distributed firewall" pose a threat to the effectiveness of network intrusion detection? Can distributed intrusion detection be implemented as well?

Some different scenarios where the "distributed firewall" (with intrusion detection) comes to rescue are explained in Chapter 5.

In chapter 6, the design and implementation of the "distributed firewall" is covered.

Some implementation issues, like what authentication mechanism and encryption method to use, are discussed in chapter 7.

Some real world usage and testing of the prototype is covered in chapter 8.

Since the start of this master thesis, other commercial and open source implementations have emerged. They are covered in chapter 9.

Conclusion and further work are covered in the last chapter 10.

Chapter 2

Background

"When speaking of computer systems, never use the word 'secure'."

— Donald H. Rumsfeld, former U.S. Secretary of Defense

2.1 Security

2.1.1 Defining security

What basically *is* computer security? The literature does not seem to agree on that definition: Dieter Gollman define computer security in "Computer Security" [28] as something that *"deals with the prevention and detection of unauthorised actions by users of a computer system"*. The U.S. Department of Defence define it in [23] as *"The protection resulting from all measures to deny unauthorized access and exploitation of friendly computer systems"*. Another definition is found in RFC2828 [63] (also called COMPUSEC): *"Measures that implement and assure security services in a computer system, particularly those that assure access control service"*. William R. Cheswick co-author of the famous firewall book "Firewalls and Internet Security: Repelling the Wily Hacker" [14] has a more bluntly definition: *"Broadly speaking, security is keeping anyone from doing things you do not want them to do to, with, or from your computers or any peripherals"*.

Even if there are several definition of computer security, the most common definition used is the one found in RFC2828 [63] given above (COMPUSEC). What *is* clear is the distinction between host security and network security.

Host security usually covers all the security mechanism in *one* computer system. Network security is used on those security mechanism needed to ensure a *secure communication* between computer systems.

2.1.2 Security service - the building blocks

RFC2828 [63] defines this as a *"service that is provided by a system to give a specific kind of protection to system resources"*. The three main "specific kinds of protection" are confidentiality, integrity and availability. Other services include authentication, access control, and non-repudiation.

"Computer security rests on confidentiality, integrity and availability" says Bishop in [8]. That does not

give any meaning unless these words are interpreted. Again, the literature has not always reached a consensus.

Confidentiality

The military was some of the first to take interest in computer security. There was need for confidentiality or a way to enforce a "need-to-know" principle. Various security mechanisms were used to conceal information or resources. This has led to the confusion that computer security *is* confidentiality.

Not only concealment of data, but resource hiding is an important part of confidentiality. Knowing that something is happening is often enough: For example a passive eavesdropper behind enemy lines may conclude that an attack is eminent due to the fact that there is an increase in communication on the military network. In Kahn's history of cryptology [42] traffic analysis played an important role before second world war: *"[...] since military operations are usually accompanied by an increase in communications, traffic analysis can infer the imminence of such operations by watching the volume of the traffic."* It could also *"deduces the lines of command of military or naval forces by ascertaining which radios talk to which"* [42]. Traffic padding may hide this information flow analysis. Bishop [8] defines confidentiality as *"the protection of transmitted data from passive attacks."*

The X.800 standard [36] has a more precise definition: *"The property that information is not made available or disclosed to unauthorized individuals, entities, or processes."*

The use of encryption is the most used security mechanism to enforce confidentiality. All communication between the master and client are encrypted and therefore providing confidentiality.

Integrity

While confidentiality has played an important role in the commercial business (keeping information hidden from competitors), it has not been the main focus. Integrity was important; to prevent unauthorized changes of information. For example, a firm is specializing in gathering sensible statistical data about persons. If the firm is exposing some data about one person to the public (confidentiality breach), it sure would be embarrassing to the firm and may cause someone to become angry. But if the firm is selling the statistical data to customers who use this in various computations, and the data suffers from random unexpected errors, it can be devastating: The firm may lose all it customers since the data is "unreliable".

To be able to trust the data (or products), there must be some sort of integrity check to either prevent or detect unauthorized or accidental changes (or both). Integrity is an important aspect of communication protocols; both IP and TCP have a header checksum to verify the packet integrity. Bishop arguments that integrity includes "data integrity", which deals with the content of the data, and "origin integrity", which deals with the origin of the data, often called authentication (although the X.800 specifies that these two terms should not be mixed).

I find dealing with integrity more difficult than dealing with confidentiality; the confidentiality is either broken or not, but it may not be so easy to see whether the integrity has been breached. There must be a distinction between integrity *prevention* and integrity *detection*. Integrity prevention tries to block unauthorized attempts to change data (using access control or authentication), while integrity detection tries to detects whether such an attempts is successful.

RFC2828 [63] has an improved definition over X.800 about data integrity: *"The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner."*

The encrypted communication between the master and client is encrypted, and any change to the cipher stream would render the decrypted plaintext garbled and produce an error. So the encryption mechanism provides integrity prevention and detection on all messages exchanged. This would be an attack on both integrity as well as availability, since the destination is unable to read the data received. In addition, the implemented host based intrusion detection tool PIDE (described in section 6.2 on page 47) tries to detect whether the integrity has been breached by doing integrity detection.

Availability

Availability is *"the property of being accessible and useable upon demand by an authorized entity"*, X.800. The Internet has become critical to business. Companies must be connected to their internal networks as well as to the rest of the world to link with customers and often their own employees. If the company's resources are unavailable, the employees may not be able to do their work and customers may not purchase products. *"An unavailable system is at least as bad as no system at all"* [8].

One of the most command and famous security attacks are on availability: DoS attack. Read more DoS attacks in section 2.2.2 on page 13.

Authentication

"Authentication is the binding of an identity to a subject" [8]. It is not always a user (subject) that needs to be authenticated; it can be any peripheral in need of authentication. The X.800 doesn't use the word "authentication", but is specified in two types: "peer entity authentication" and "data origin authentication". Peer entity is the confidence that the identity of the entity is the one claimed. This prevents masquerade (spoofing), replay or man-in-the-middle attacks. Data origin is assurance that the source of the data is the one claimed.

The use of authentication is especially important when the management node is contacting the clients and vice versa. If the authentication is compromised, an attacker may connect to any client and enforce some malicious command.

Access Control

Access control is a restriction to prevent and control usage of resources. To gain access, the user must first be authenticated and, if the user is allowed, given authorization (access).

The "802.1X Port-Based Network Access Control" [34] provides such an access control. All wireless nodes must first be identified before gaining access to other LAN resources or the Internet. 802.1X is used in the new wireless security standard "802.11i Medium Access Control (MAC) Security Enhancements" [35] which should replace the flawed WEP encryption used in (old) wireless equipment. Read more about both 802.1X and 802.11i in my paper [46].

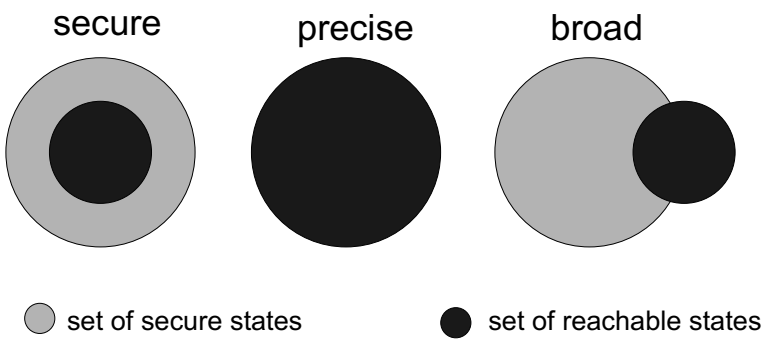

Figure 2.1: Types of security mechanisms.

Non-repudiation

To be able to proof that a party really did participate in a communication, is called non-repudiation. Either nonrepudiation by origin; the message was in fact sent by the specified party, or non-repudiation by destination; proof that the recipient received the message.

2.1.3 Security policy

While security services give a specific kind of protection to a set of system resources, the security policy often consists of several security services. While RFC2828 has a more specific definition of what a security policy is, it basically says the same as Bishop [8] *"A security policy is a statement of what is, and what is not, allowed."*

A security policy can be expressed in mathematical terms or in plain English. It defines when a system is in a "secure" state and what constitutes a system in an "insecure state". A security policy may be more specific: A confidentiality policy, also called "information flow policy", deals with the exchange of information. The most famous confidentiality policy is "The Bell-LaPadula Model" [19] and [20] which conforms to military style classifications. Commercial requirements were often different than military; the data/products offered had to be trusted. A random or erroneous change to a users bank account could be devastating to the banks reputation. Integrity policy deals with the integrity and puts trust to the data. The most famous integrity policy is "Bibas Integrity Model" [7], "Lipner's Integrity Matrix Model" [47] and "Clark-Wilson Integrity Model" [15] all explained in detail in [8].

2.1.4 Security mechanism

A security mechanism is policy implemented. It ensures that the policy is obeyed. Stallings [66] define a security policy as *"a mechanism that is designed to detect, prevent or recover from a security attack"*.

While a security service can be pretty straightforward, often expressed in on-word statements; "confidentiality", "integrity" etc., the security mechanisms implementing a security service can be pretty complex.

A security mechanism may not always fulfil a security policy; a mechanism is secure if all the reach-

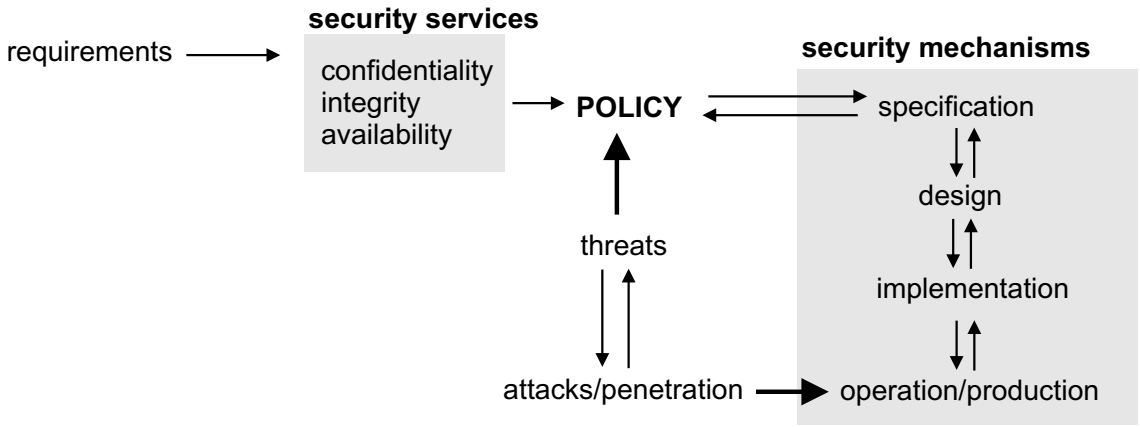

Figure 2.2: The ever-evolving process of security

able states are in the set of secure states. I belive security mechanisms protecting a corporate network are seldom/never secure; there is often reachable states that are not secure. If all reachable states are equivalent to the secure states, it is considered precise. Again, this state is rarly reached. A broad security mechanism does cover some, but not all, set of reachable states. See figure 2.1 on the facing page. The firewall tool implemented in this master thesis implements a "broad" security mechanism.

This master thesis implements a security policy, using several security mechanisms, that prevents many of the security attacks that the traditional firewall doesn't cover (by using a distributed firewall). It also detect if an attack was successful (by using host based intrusion detection).

If a security mechanism fails to implement a security policy, it does not justify exploiting the failed mechanism. To use an analogy; if the owner of a house has failed to lock the door, does not justify a thief walking in. A security mechanism may also prevent one ting, but be used in the wrong way to do malicious things by others. The famous "wizard" password in Sendmail years ago illustrate this; the author of Sendmail implemented a `WIZ` command to sendmail to enable non-standard SMTP commands, including giving a (root) shell. The system administrator (or attacker) could just telnet to port 25 (SMTP) on the mail server, type in `WIZ` and `SHELL` and would be given a root shell[1] By using the "wizard" mode, the attacker was given a free-shell to the mail-server to do malicious actions.

2.1.5 Security architecture

As seen on figure 2.2 the security process is a never ending process of new threats/attacks which may change the policy which in turn affects the security mechanism and design of the finishing product.

An simple way to evaluate and calculate risk are by using the formula $threat \times vulnerability \times value = risk$. Several more extensive standards have emerged over the years that provide a professional evaluation of computer security. The United States developed the "Trusted Computer System Evaluation Criteria" (TCSEC) [68], also called the "orange" book, early in the 1980's. In the following years, other countries developed other similar standards; Canada developed the "Canadian Trus-

[1]The Sendmail author did intend that the user had to provide a password to the WIZ-command given in the Sendmail configuration file (on-way hash immediate followed after 'OW' in Sendmail's configuration file). But due to a programming bug, the password was set to NULL after the configuration files was "frozen" (e.g. malloc dumped to file) and Sendmail re-ran. Read more here: `http://groups.google.com/groups?hl=en&lr=&selm=CpLAD6.41E\%40ulysses.homer.att.com`

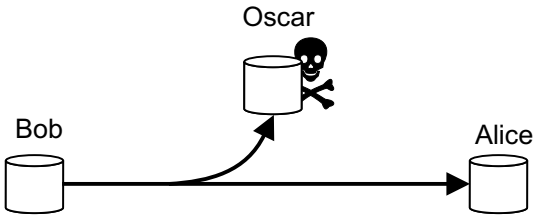

Figure 2.3: Release of message content/traffic analysis.

ted Computer Product Evaluation Criteria" (CTCPEC) [11], based on TCSEC in 1989. The European Union published "Information Technology Security Evaluation Criteria" (ITSEC) [17] in 1991.

In June 1993, a joint effort to create a common criteria was begun. The goal was to create one single set of security criteria that could be used. The official name of these criteria is "Evaluation Criteria for Information Technology Security" [12], but often just called "Common Criteria" (CC). Version 1 of CC was completed in 1996, and the current version 2 in 1997.

2.2 Threats and Attacks

2.2.1 Threats

A threat is *"a potential violation of security"* X.800 [36]. A threat is not dangerous in itself, only when an attack is launched against it and causes harm.

2.2.2 Security Attacks

A security attack is an *implemented* threat: *"A specific formulation or execution of a plan to carry out a threat"* [3], or more specific: *"An assault on system security that derives from an intelligent threat"* RFC2828 [63]. Successful attacks are often designed by looking at the problem in a completely different way.

There is a distinction between inside and outside attack. An outside attack is an attack initiated from the "outside" the security perimeter and an inside attack from the "inside". This distinction between outside/inside topology is beginning to fade as nodes become more mobile, which is one of the main purposes of this master thesis. More importantly are the distinction between active and passive attacks.

Passive

A passive attacker does not alter the system resources, but learns from the information leakage. This kind of attack is difficult to detect, so the main focus is to prevent.

Release of message content Two passive attacks exists, both of which can be prevented by use of confidentiality. As seen on figure 2.3, Oscar can read the message Bob intended to Alice. With the expanding usage of wireless devices, security mechanisms to prevent passive attacks are becoming more important. With wired networks, the evader must have physical access to the network to do

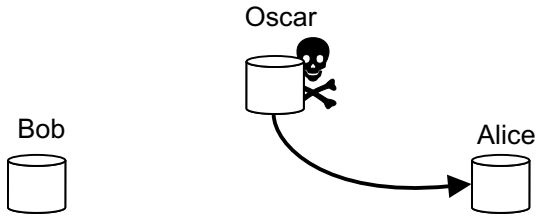

Figure 2.4: Masquerade. Message from Oscar appears to be from Bob.

passive attacks - with wireless networks, just being within radio range is enough. The attack on the badly designed WEP-encryption[2], used by all (old) 802.11 devices, by using the famous WEP-cracking tools Airsnort[3] or Kismet[4], is passive. These tools just listen for interesting packets (packets using the same IV) without any active interference.

Traffic Analysis Even if Bob and Alice were using some sort of encryption to prevent Oscar from reading the message, Oscar may learn that the message was in fact sent, see figure 2.3 on the preceding page. This might be enough for Oscar: If Bob is a general and Alice an officer of an army, Oscar may conclude that Alice has been given orders to go to attack. This is why military networks often uses traffic padding to *"fill up the pipe"* with garbled traffic.

Active

Active attacks are more easily detected because these sort of attack often leave behind some trails. The majority of all security attacks are active.

Masquerade Also called spoofing, is when a user tries to take the identity of another. In figure 2.4, Oscar impersonates Bob. It need not be masquerading of users; it can be masquerading of IP-address, hostname, MAC-address etc.

It is often important for the attacker to use spoofing. For example: To find out what sort of services are running on a remote host, a user may telnet into the respective TCP port. Unless a 'connection refused' is received, the port is most likely open. For example: To check whether a mailserver (SMTP) is running on a host, a user may telnet into port 25 of a host:

```
> telnet some.host.here 25
Trying 123.123.123.123...
telnet: Unable to connect to remote host: Connection refused
```

The host `some.host.here` does not have a mailserver running on port 25. Running the same command against a different host:

[2]The new 802.11i [35] security standard, which was ratified in June 2004, fixes all WEP weaknesses and implements a Robust Secure Network

[3]Airsnort homepage: `http://airsnort.shmoo.com/`

[4]Kismet homepage: `http://www.kismetwireless.net/`

```
> telnet some.host2.here 25
Trying 123.123.123.124...
Connected to 123.123.123.124.
Escape character is '^]'.
220 some.host2.here ESMTP Postfix
```

Here the user is being "logged" into the port and the mailserver is accepting SMTP commands [25]. The problem, for an attacker by using a this kind of "service discovery", is that it is time consuming to check all ports, and it leaves traces. A careful examination of the mailserver logs reveals the hostname and IP address of someone tried to connect:

```
Aug 29 19:06:02 trinity postfix/smtpd[80681]: connect from some.evil.attacker
```

To prevent leaving such an obvious trace, more effective ways of probing is used. The most common is "port scanning", which probe a host for open TCP ports. There are several different techniques to do a portscan, but they all rely on the response from an open TCP port. Normally an open TCP port responds to a new TCP request with a TCP packet with the SYN and ACK bit set[5]. A closed port responds with a TCP packet with the RST bit set. The sequence of the different TCP packets crafted may make it harder to detect that the host is being probed. The most famous port scanning tool is Nmap[6] which provides a wide range of portscans including Xmas-, FIN-, NULL-scan, all which are sending different variants of TCP-bits set in different order. An example of a Xmas portscan using nmap:

```
> nmap -sX localhost

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-08-29 19:37 CEST
Interesting ports on localhost (127.0.0.1):
(The 1652 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
9/tcp   open  discard
13/tcp  open  daytime
22/tcp  open  ssh
25/tcp  open  smtp
37/tcp  open  time
111/tcp open  rpcbind
631/tcp open  ipp

Nmap run completed -- 1 IP address (1 host up) scanned in 4.141 seconds
```

Even if a portscan may not leave any obvious traces behind, it may very well be detected by a sensitive Network Intrusion Detection System (NIDS). More advanced portscans, like "bounce scans", does work with help of a third party host. These attacks are masquerade attacks, and make the job of revealing who the real attacker is more difficult.

[5]Both the TCP specification [33] and RFC1180 TCP/IP tutorial [65] contains more information
[6]Nmap homepage: `http://www.insecure.org/nmap/`

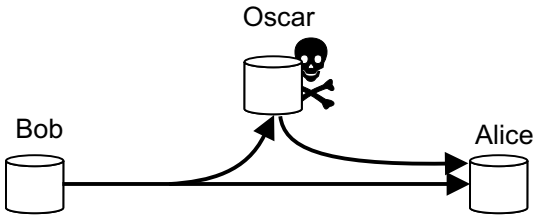

Figure 2.5: Replay. Oscar capture message from Bob destined for Alice. Oscar later replay message to Alice

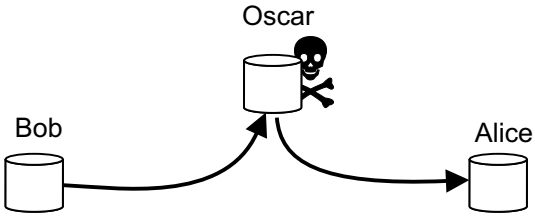

Figure 2.6: Modification of Message. Oscar modifies message from Bob destined to Alice. Also called "man-in-the-middle" attack.

Replay Occurs if one entity captures message destined to another entity, and later resends it. As seen on figure 2.5, Oscar captures message from Bob to Alice and later resends it.

If Bob and Alice where routers/gateways exchanging BGP [56] routing information between Autonomous Systems (AS), the replayed message from Oscar may contain false updates. A false BGP message may make the routing information within an AS false and thus making hosts, or in worst cases; AS's unreachable. This is a clear attack on availability.

Modification of Message Occurs when Oscar intercepts message from Bob destined to Alice, modifies it, and sends it to Alice. The attacker must be able to listen to and intercept messages between Bob and Alice. This is also called "man-in-the-middle" attack. See figure 2.6.

This is an critical threat to the usage of public key based cryptosystems; the communicating parties (public) keys must be known and trusted beforehand. If not, the parties are vulnerable to man-in-the-middle attacks. The problem of distributing keys are discussed in section 7.6 on page 76.

Denial of Service DoS for short, is probably the most common and most frequent security attack[7]. DoS attacks are usually remotely and used to overwhelm the targeted system with bogus network packet, thus rendering the service unavailable. A DoS attack may also be used to crash the remote system by exploiting a software bug, either locally or remote. The losses caused by security attacks, according to the annual CSI/FBI report [29], are estimated to $26 million dollars, only bypassed by virus. A distinction can be made:

- **Flooding** can be either local or remote:

[7]The CERT description of DoS attacks is found here: `http://www.cert.org/tech_tips/denial_of_service.html`

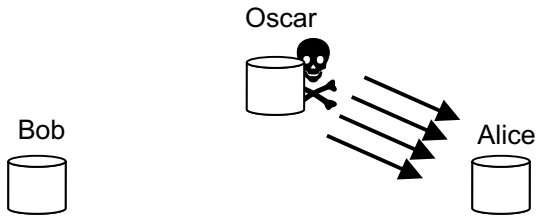

Figure 2.7: Denial of Service. Oscar sends large amount of bogus traffic to Alice which consumes all Alice's resources. Often used "distributed" (DDoS); more than one host DoS Alice at the same time.

  - **Local DoS attack** is not an attack using network, but exploiting a software bug to consume the hosts local resources (memory, CPU, disk, .. etc.). For example: Creating a program that does a `fork` within a never ending `while` loop, consuming all available memory is a local DoS attack.
  - **Remote DoS attack** involves a huge amount of packets sent over the network, consuming all available bandwidth and making the service offered unavailable to real requests.

- **Malfunction** is specially crafted packets that exploit certain software bugs that crashes the running service or, in worst case, the whole operating system[8].

There are several problems with DoS attacks that make them so effective. By using spoofed source addresses, the attacker may hide his origin. From the targets point of view, the attack may come from "anywhere". It can be easy to amplify the attack, by using ICMP echo (ping) packets to broadcast addresses (which makes all hosts on the network answer). If the source address, of the ping packet, is spoofed to the target hosts, the targeted machine would receive all the ICMP reply (pong) messages. Using broadcast to amplify the attack is called "smurf" attack[9].

The most "popular" attack among cracker, is a variant of DoS, called *Distributed* DoS (DDoS) attack. The reason why DDoS are popular because they are simple and effective.

A DDoS attack is basically a large amount of nodes, each sending a vast amount of packet with spoofed source address to a target. The nodes which participates in DDoS attack are very often been "taken over" or compromised by an attacker. The compromised hosts are also called "zombies" [10]. All the network traffic combined is too much for the target to handle, leaving the targeted system/service unreachable.

The last, and most important issue containing DoS attacks, is that there is no clear defence mechanism. Since the sources of the attacks are spoofed, the only way of tracking a zombie is to investigate all hops (routers/gateways) to find the real source. This task may seem as a nightmare since it involves contacting all administrators of the routers/gateways along the way to the source. And if a site is being DoS by several hundred different zombies from all over the world; the task may be close to impossible. One solution to stop DDoS attacks is to do network ingress filtering as described in

[8]The famous "ping of death" was such a specially crafted ICMP packet. The attack was not particularly intelligent; most network stacks in operating system at that time (1996), did not like ping packet larger than 65536 bytes - making the whole operating system freeze. Read more here: `http://www.insecure.org/sploits/ping-o-death.html`

[9]Read more on the original CERT Advisory: `http://www.cert.org/advisories/CA-1998-01.html`. A list of networks that may function as smurf amplifiers can be found here (it can be a good thing to block ICMP traffic from this database): `http://www.powertech.no/smurf/`

[10]A zombie is host which has been "infected" by software that may take commands from a cracker. A cracker may control hundreds of these zombies and may initiate a DDoS attack by issuing commands like "SYN flood target X".

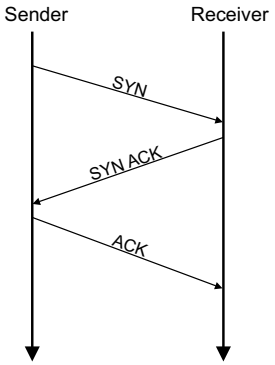

Figure 2.8: "TCP three-way handshake".

[26]. The problem to make this work is that a large number of service providers must employ ingress filtering at their routers/gateways[11].

TCP SYN-flood attack[12] is perhaps the most "arch-typical" DoS attack and has been used for several years [13], and so has the tools for crafting these attacks [18]. SYN-flood exploits the design of the TCP handshake: The TCP handshake is three way, as seen on figure 2.8:

1. Node A first sends a TCP packet with the SYN (synchronize/start) bit set and a sequence number.

2. B responds to A's request with the SYN and ACK (synchronize acknowledge) bit set, including A's original sequence number + 1, and B's sequence number.

3. A then confirms B's response by sending a packet with the ACK (acknowledge) bit set and B's sequence number + 1, back to B.

For more details, read the TCP specification in RFC793 [33].

The SYN-flood attack is carried out by sending a vast amount of TCP SYN packets to the target node. The source address is spoofed, and the spoofed addresses are unreachable, so the targeted host will have "half-opened" connection until they time out. When a large amount of these connections are made, the resources available are consumed, and new legitimate connections are refused.

One solution[13], although its not a *solution*, it merely makes the targeted host more *resistant* to withstand the attack, is called "SYN-cookies"[14]. SYN-cookies generate the sequence number by making a hash of some values (secret, source address, destination port, ...). Since this sequence number must be acknowledged, no state needs to be saved. When (if) a returning ACK arrives, the returning sequence number (minus one) is compared to the hash value (which changes every 4th second - a time set as reasonable upper bound for RTT). Since no state is saved, there are no "half-opened" TCP

[11]SANS Institute have created a "Consensus Roadmap for Defeating Distributed Denial of Service Attacks" found here: `http://www.sans.org/dosstep/roadmap.php`

[12]The original "CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks": `http://www.cert.org/advisories/CA-1996-21.html`

[13]Other defences against SYN-flood include "micro blocks", "RST cookies" and "stack tweaking". None which are in common use today.

[14]"SYN Cookies" are credited to D. J. Bernstein `http://cr.yp.to/syncookies.html`

connections. Even if the target uses SYN-cookies, a SYN-flood attack may still be successful if the flooding is large enough[15].

Starting to take greater care in protecting computers from take-overs, is certainly a good start. But, as John Earl from The Power Tech Group says: *"Security is the responsibility of the TARGET system"*. Perhaps the only solution to effectively stop these attacks are by implementing what Steven M. Bellovin calls the "evil bit"[16] in the IP header?

2.3 Cryptography

The word "cryptography" is a combination of two Greek words (*kryptos* and *graphos*) meaning "secret writing". The roman emperor Caesar is one of the first known to use cryptography; by sending orders using a monoalphabetic substitution cipher[17]. Having only enough patience, the cipher can be deciphered easily. But at the time, it was enough: The orders where unreadable and created confusion, unless the correct numbers of substitution was used.

Today cryptography is a mathematical subject, using computers to crunch computationally intense algorithms. Cryptography is an important security mechanism to enforce security policies today: Banks, financial institutions and e-commerce rely on cryptography to enforce confidentiality. Public Key encryption can be used to enforce authentication, integrity and confidentiality. One-way hash can be used to ensure integrity of data, and even authentication, if used with a secret key.

2.3.1 Encryption

To implement security mechanisms, some sort of cryptography is often used. The key to security is not "security to obscurity", but rather the trustworthiness of the algorithm used. Or as Bruce Schneier put's it in [61]: *"Trying to base security on secrecy is just plain bad design."* A good cipher algorithm protects against the following attacks:

- **Ciphertext only attack:** Only the ciphertext and the encryption algorithm used are known. This makes an attack very difficult; even if the algorithm can be broken within a reasonable amount of time - the plaintext can be anything. Is the plaintext in English? In Norwegian? Even harder if the plaintext is a binary file. For each tried key, the message must be examined carefully.
- **Know plaintext attack:** The same as *ciphertext only attack* is known, in addition to the plaintext. This attack can be used to find the encryption key used.
- **Chosen plaintext:** The same as *known plaintext attack* is known, except that the plaintext is chosen by the attacker. Some early wireless Access Point (AP) accepted both unencrypted and encrypted broadcast. When an unencrypted broadcast was sent, the AP sent the broadcast both encrypted and unencrypted. This made it even easier to crack the poorly designed WEP encryption.

[15] Unfortunate, SYN-cookies are not enabled by default under Linux. To enable (assuming support is compiled into the kernel): `echo 1 > /proc/sys/net/ipv4/tcp_syncookies`

[16] In [4] published 1. April 2003, the "evil bit" is using the only unused IP bit in the IP header, which is the high order of the IP fragment offset, to distinguish between "evil" and non-evil (or normal) packets.

[17] This cipher, also called "Caesar cipher", is used by mapping each letter three characters later in the alphabet. The plaintext "Crypthography is fun" would produce ciphertext: "Fubsxrjudskb lw iyq"

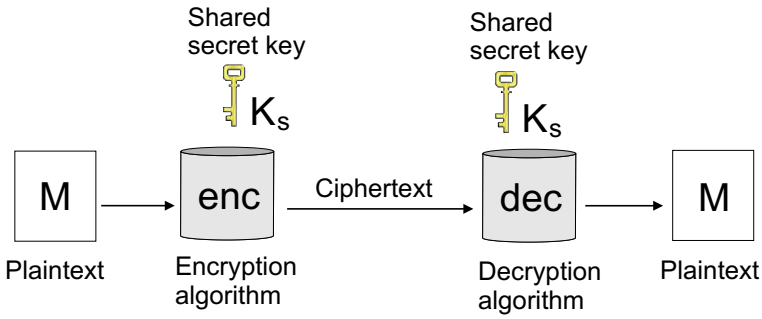

Figure 2.9: Basic outline of symmetric encryption. The same shared key is used to both encrypt and decrypt.

According to Stallings [66], to make an encryption scheme computational secure, the ciphertext must meet the one or both of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

A good algorithm should only be broken by exhausting all possible keys. This form of attack is called "brute-force". Since the time to brute force an algorithm grow exponentially with the key length, newer algorithm with long keys take too long to brute-force. That's why attacker tries to find "short-cuts" to break the algorithm; which usually is exploits in implementation of the algorithm.

There is an important distinction between symmetric and asymmetric encryption:

Symmetric encryption

The basic concept of encryption is to feed an encryption algorithm with the plaintext M, the message to be encrypted, and something secret, the key K_S, into an encryption algorithm. The produced effect is a ciphertext, see figure 2.9.

Symmetric encryption uses a shared secret key to both encrypt and decrypt. The secrecy of symmetric encryption relies on holding the shared secret key secret, not the secrecy of the algorithm used. Since many of these algorithms are open, they can be implemented in hardware and have gained wide-spread use. Symmetric encryption is faster than asymmetric, but there is no authentication since the key is shared. DES and AES is the two most used symmetric encryptions.

Data Encryption Standard (DES) The most common and most used symmetric encryption used is DES [51]. Becoming a U.S. national standard in 1977, the 56-bit key length was more than enough at that time. Still to this day, no single publicly acknowledged weakness exists in DES. Today a 56-bit key is to small, as in 1998 a special purpose machine was built by the Electronic Frontier Foundation (RFF) to brute force DES. The attack on DES took three days.

3DES was an improvement of the original DES with a key length of 168- or 112 bits. It basically is DES done three times: Encrypt message with one key, decrypt again with the second key and encrypt the last time with the third and last key. If all the three keys are the same, it is compatible with the

original DES. This new key length should prove secure, or as Bruce Schenier says: "there isn't enough silicon in the galaxy or enough time before the sun burns out to brute-force triple-DES" [59].

Unfortunate, DES is slow and uses a block size of only 64-bits. All of these shortcomings are met in the new Advanced Encryption Standard (AES). In July 2004, AES replaced DES as official encryption algorithm[18]

Advanced Encryption Standard (AES) AES [39] is the new encryption standard that will eventually take over for DES/3DES. AES uses a block size of 128 and has a key length of 128, 192 or 256 bits. No know attack on the algorithm is known. AES can effectively be implemented in hardware and can have high throughput.

Asymmetric encryption

Asymmetric encryption, also called public-key encryption, *"is the first truly revolutionary advance in encryption in literally thousand of years"* [66]. It uses two keys; one key to encrypt and another to decrypt. One key is private, and never shared with anyone. The other key is public and is shared with everyone who wants to communicate securely with the user.

Asymmetric encryption is not more secure than symmetric, and it will not obsolete the symmetric encryption. Asymmetric is slower than symmetric and therefore not so attractive when it comes to performance.

If B wants to send user A a private message, B uses A's public key to encrypt the message. The message can then only be decrypted using A's private key. The requirement for making this scheme work is that only A is in the possession of the private key. There is also a challenge to distribute the public key and trust that the public key really is from the claimed user. PKI, as described in section 7.6 on page 76, tries to solve this problem.

RSA was one of the first asymmetric algorithms, and still is one of the most famous. Published in 1977 and named after its inventors: Ron Rives, Adi Shamir and Len Adleman. RSA may be used in "Pretty Good Privacy" (PGP) which provides confidentiality and authentication service to mail.

Modes of operation

A mode of operation is how the data is processed before it is encrypted/decrypted by a cipher algorithm. A symmetric block cipher process one chunk of data at the time, called a block. A block of data may depend on the cipher being used. DES uses a block length of 64-bits while AES uses 128-bits.

If each block of data is being encrypted with the same key, all blocks of equal plaintext produces an equal pair of ciphertext. This mode is called Electronic Codebook (ECB) and should not be used on messages larger than one block. This makes an attack easier, since it may be easier to guess the plaintext; for example the start of a HTTP request starts with "GET" and a link, and gives the attacker the possibility to perform an "know plaintext attack" as described above.

Other modes, like Cipher Block Chaining (CBC) uses the XOR of the produced ciphertext of one

[18]The U.S. Government is officially withdrawing DES as an encryption standard: `http://csrc.nist.gov/Federal-register/July26-2004-FR-DES-Notice.pdf`

block, as IV into the next. So even if two blocks of plaintext are the same, they will produce different ciphertext.

A stream cipher does not need to pad the message up to the nearest block size, and may operate in real time. A block cipher may be converted into a stream cipher by using Cipher Feedback Mode (CFB). The disadvantages by using a stream cipher are that its slower than a block cipher.

I'm using AES encryption, with CBC-mode, using a 128 bits key to protect the PIDE database. This is done in *blackbox*, as described in section 6.3 on page 53.

2.3.2 Secure Hash

A one-way hash function creates fingerprint of a given message. Changing even the smallest portion of the message should produce a totally different fingerprint. The hash function most used today is MD5 [57] and SHA-1 [24]. For a hash function H to be useful for message authentication, Stallings [66] lists the following properties:

1. H can be applied to a block of data of any size
2. H produces a fixed-length output.
3. $H(x)$ is relatively easy to compute for any given x, making both hardware and software implementation practical.
4. For any given value h, it is computationally infeasible to find x such that $H(x) = h$. This is sometimes referred to in the literature as the one-way property.
5. For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$. This is sometimes referred to as weak collision resistance.
6. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. This is sometimes referred to as strong collision resistance.

2.3.3 Message Authentication Code (MAC)

A MAC is an authentication "tag" appended to a message. The receiver computes his own tag and compare it with the recived one. If they are equal, the message is authentic. There are basically three ways to generate a MAC:

1. By using symmetric encryption with a shared key on the message. Only a portion of the encrypted message is used as the MAC, usally the last 16- or 32 bits part of the ciphertext.
2. By using a one-way hash on the message and a shared secret, the hash result functions as a MAC.
3. Asymmetric encryption may also be used; by first creating a one-way hash of the messages, the hash result is then encrypted with the receivers public key. Only the holder of the corresponding private key can be used to decrypt the hash result.

Traditionally symmetric encryption has been used to generate MACs. The "National Institute of Standards and Technology" (NIST) recommend using DES in "Processing Standards Publication (PUB)" 113 [50]. But there are several problems using encryption as MAC;

- Encryption is slow compared to one-way hash functions.
- Encryption is often optimized for large chunks of data. Computing a MAC using encryption on small messages produces too much overhead in initialization.
- Encryption algorithm may be covered by patents and export control.

So by using on-way hash functions instead, the MAC is computed faster, and is not restricted by export control (in the U.S.). Using one way-hash functions when computing MAC, is also called HMAC [44] and is being used in other protocols as Transport Layer Security (TLS) [22] and IPSec[19].

[19] IPSec charter: `http://www.ietf.org/html.charters/ipsec-charter.html`

Chapter 3

Firewall technology

> "If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology."
>
> — *Bruce Schneier*

A firewall is a pretty broad term, which may include many different levels of protection and technology. In [14], firewall is defined as *"[...] any device, software, or arrangement or equipment that limits network access"*. Or put more bluntly, from "Building Internet Firewalls" from O'Reilly [73]: *"If it's supposed to keep the bad guys out of your network, it's a firewall."*

3.1 Firewall is a good thing

There are several reasons why the use of firewall has gain such a widespread use. The main reason, I belive, is because it is such an easy way to enforce security policy. By putting a firewall at the gateway of the local network, the internal hosts are protected from malicious attacks from the outside.

Firewall is also an "old" protective mechanism [49], especially packet filtering firewalls. It has been implemented in a large number of operating system and are one of the most used security technology used, used by 98% according to the annual CSI/FBI survey [29], only beaten by antivirus software. The corresponding Norwegian report [74], has detected an increase of firewall usage from 67% in 2001 to 84% in 2003. It is a technology that has got a lot of research attention, and is a technology that has been "well-proven" in real-life.

3.1.1 Simplicity

A firewall is a bottleneck that all traffic to and from the outside must pass through. Instead of having to spread all the security mechanisms onto all the hosts, which may pose a maintenance nightmare, there is just one focus. This makes it easier to maintain; the gateways (firewalls) are the only host requiring attention.

For example, the usage of peer-to-peer (P2P) traffic is increasing rapidly. A poorly designed P2P program may share out all the files on the local computer. This can be a serious breach of the security policy. To prevent this from happening, the firewall may block all incoming and outgoing P2P traffic.

3.1.2 Buggy programs

Most of the security bulletins posted on CERT[1] and others, are regarding buggy programs. Many of these are again buffer-overflows which may not only crash the program, but in worst cases spawn malicious code[2]. Keeping track of all the hosts on the inside and make sure they all are up-to-date on the latest security patches, may be too much to ask for a system administrator. A firewall protected all these hosts.

But it is *important to remember that internal network does not become more secure because of a firewall.* The scenario gets worse when private laptops are plugged into the corporate network; are they patched against the latest security holes? A compromised laptop may pose a security threat when plugged into the corporate network. This is one of the main reasons I've implemented a distributed firewall; to protect the internal network.

3.1.3 Older protocols/programs not designed for security

Many older programs and protocol are not designed with security in mind. For example the "Network File System" (NFS) [62], which allows systems to access files across the network on a remote system, has a dubious security history. To redesign and reimplement these, to support strong cryptography (authentication, confidentiality etc.) may not be an option; it will take to much time and resources. A firewall protects these systems from outside attacks.

3.1.4 New threats

Even though a firewall can not *detect* new threats, which is the work of an intrusion detection; it can block potential new threats. For example, when the Windows RPC (MS03-026[3]) worm hit the Internet August 11 2003, it shut down unpatched Windows hosts. It was so effective that a system administrator could not do a clean install of Windows, head over to Windows update for patches, before the host was hit by the worm. The hosts had to be installed using a pre-patched image or patch the Windows host off-line. Hosts that were behind a firewall that blocked the used RPC port (usually 135, 139 and 445) were unaffected.

3.1.5 Logging

A good firewall should always provide extensive logs of the network traffic. But the logs are useless if they are not examined: *"Having a firewall set up on your system doesn't do you much good if you don't monitor it for abnormal behaviour"* [60]. When an intrusion has occurred, the (firewall) logs should be the first place to start. But the logs are also the first thing an attacker goes for to try hide the break-in.

[1] Computer Emergency Readiness Team Coordination Center (CERT/CC): `http://www.cert.org/`

[2] In C and C++ some input data, if not the input data is checked thoroughly, may write past the bounds of the buffer. A cracker may craft a buffer so that, for example the return pointer of a function, may return to the input data itself. This data, that contains malicious code, may spawn a shell or open a back-door. CERN 1998: 9 out of 13 security bulletins was buffer overflows. For more information see `http://www.linuxjournal.com/article.php?sid=6701` and `http://www.phrack.org/show.php?p=49&a=14`

[3] Read more about the vulnerability and exploit here: `http://www.cert.org/advisories/CA-2003-19.html` and `http://support.microsoft.com/?kbid=823980`

The logs should therefore be attack-free. Either by some sort of integrity or encryption mechanisms. It is also common to log to another host (a "log-server")[4].

3.1.6 Limit exposure

Even though Emmanuel Goldstein claims *"There is no such thing as security through lack of information."* I believe this is partially true when it comes to protecting the internal network; why give away the topology for free? Some of the first information an attacker tries to gather, is the network topology of the targeted network. Once he has the topology approximately figured out, with a few hosts exposed, he launches his attack.

A decent firewall should not give the attacker the opportunity to obtain the topology easily; by either pinging the IP-range of the network in question, or do a "port-scan". Often an attacker starts a "port-scan" on very large network-ranges (typically a B-net[5]), and returns days later to parse the port-scan logs looking for interesting hosts.

3.2 Firewall limitations

A (packet filtering) firewall have been in common use for over two decades. There are several shortcomings of the traditional firewall. Today, several new technologies are emerging that doesn't cooperate well with the traditional firewall. Some shortcomings are more important:

3.2.1 Maintenance

There is only very small corporate network that can rely on only *one* physical firewall. When the size of the network and the traffic increases, additional firewalls are often used. There can be special purpose firewalls that has been given special tasks; one to filter HTTP traffic, another to filter packets. Not only are the numbers of firewalls providing a challenge for the system administrator, but the chance of configuration errors increases with added firewall configuration (number of rules).

A firewall must often contain rules that give certain hosts special treatments; some hosts may need more access than other hosts. All different types of servers, which provide different services, use different TCP-ports. Even users may need more access than other users. The configuration gets even worse if the network is using dynamic allocated IP addresses (DHCP) and the users, needing special access, are using different hosts. All these firewall "rules" quickly becomes a maintenance nightmare.

In "A quantitative study of firewall configuration errors" [70] a number of firewall configuration rule-sets was collected. The rule-set was then analyzed to see if they contained *"violation of well-established industry practices and guidelines."* The rule-set complexity, RC, was defined as:

[4]For the really security paranoid, the ethernet cable that goes into the log-server should only have pins providing traffic *in* connected. All other pins should not be connected. This way, the log-server only accepts incoming traffic, and has no way of sending traffic out. Since most remote log-services are using UDP, the log-server receives the logs just nicely. This makes the log-server very hard to compromise, since the attacker will have a hard time getting any response from the log-server.

[5]Before "Classless Inter-Domain Routing" (CIDR), covered in RFC1518 [55] and RFC1519 [27], the Internet's IPv4 address-space was traditionally split into three broad segments; called A, B and C classes. The IPv4 range contains 126 class A networks (each containing 16777214 hosts), 65000 class B networks (each containing 65,534 hosts) and 2,097,152 class C networks (each containing 254 hosts). There is also a class D, which is reserved for IP multicast addresses, and a class E, reserved for future use.

$$RC = Rules + Objects + \frac{Interfaces(Interfaces - 1)}{2}$$

Wool defined a "well-configured" firewall as one with three configuration errors or less. Only simple rule-set, with RC values under 100, could be considered "well-configured". Since, as the complexity of RC increased, the number of errors increased by $\ln(RC) + 1.5$.

The article concludes with that keeping the rule-set as simple as possible is more important. Instead of adding rules to an already complex rule-set to include for example a new sub-net, a new, dedicated, firewall with simpler rule-set is preferred. This conclusion can also be used as an argument for the use of distributed firewall: All nodes using a distributed firewall has a simpler rule-set than a traditional gateway firewall, and should therefore have less configuration errors.

3.2.2 End-to-end principle

A firewall interfere with one of the basic principles of the way Internet works; end-to-end communications. Several protocols does not work well with firewalls; like mobile IPv4 [52], and FTP [54]. The File Transfer Protocol (FTP) [54] is perhaps the most used example; FTP has two modes, "normal" and "passive". In "normal mode", the commands go over port 21, and data from port 20 (initiated from the server). Using "passive mode", the FTP server sends a port number above 1024 to the client, which the client uses to connect to and use as the data channel. The fact that the client connects to a random port on the FTP server, makes filtering difficult for the firewall.

Special firewall configuration may be applied to "work-around" these protocols, but that only adds to configuration complexity - which, as stated above, should be avoided.

But this is perhaps the price to pay; the Internet today poses too much of a threat to take the risk of exposing the whole intranet to enable end-to-end communications.

3.2.3 Internal traffic

According to the [29] report, the amount of successful attacks have been steadily declining since 2001. The report does not try to explain *why* this has been declining; whether it is because lesser attack has been launched or because of better deployment of security within the firms. I'm tempted to believe the latter; I do not think the numbers of attacks are declining. People on the Internet have not suddenly become nicer.

One explanation may be Windows 2000. The majority of desktop computers are still using Windows. As of Windows 2000, Microsoft tried to beef up the security from the previous Windows 9X release cycle. Windows 9X was a total catastrophe from a security point of view. Windows 2000/XP still has a long way to go; is not considered "secure" if it is compared to security focused operating systems like OpenBSD. It also has a high number of security vulnerabilities, with a predicted record breaking three numbered amount this year (2004). But with the arrival of Windows 2000, the desktop has become a little more secure and harder to attack. This may be a contributing cause why the numbers of successful attacks are declining. Since it does not look like the computer industry are going to replacing Windows with OpenBSD on the desktop in the near future, there will still be need for other security mechanism.

According to the [29] and the corresponding Norwegian report [74], the most used security mechanism after antivirus, is firewalls. Close up to 100% uses firewalls to protect the corporate intranet. But

the traditional "gateway" firewall has faced a number of challenges and some has even claimed the firewall an obsolete security mechanism.

Internal attack

One major drawback for the use of traditional firewall is that it does not protect against internal threats. Since the traditional firewall usually is located at the gateway to the internal network, it can't filter the inside traffic. A firewall can't filter traffic it can't see. According to the annual CSI/FBI "Computer Crime and Security Survey" [29] the number of successful attacks from the inside is roughly equal to numbers from the outside. So just protecting from outside attacks covers just half the threats.

For example: If an employee decides to set up a wireless access point of his own, to be able to roam around his office with his laptop, it will also become a backdoor into the network. Setting up rouge access points usually is a serious breach of security policy. Fortunate, if using 802.11i [35], this threat is minimized[6].

External attack

A firewall has often very strict rules concerning traffic *from* the outside destined *to* the inside. Traffic going the other way, from the inside and out, might not be protected/filtered at all. This enables an inside cracker to launch an attack from the inside to targets on the outside. If the firewall is using some sort of NAT, the cracker's identity may also be hidden, revealing only the IP address of the NAT'ed hosts. A well-configured firewall should filter both incoming and outgoing traffic.

3.2.4 Interpreting the traffic

With the increased use of end-to-end encryption (IPSec/VPN), the firewall has no way to examine the encrypted packets.

3.2.5 Single point of failure

Internet speed, and amount of traffic, is steady increasing; ordinary desktop/laptop hosts come with gigabit ethernet connection, and streaming of media (radio/TV/movies) and other Internet services increase the bandwidth usage. All of this increases the pressure of the firewall, which has to check every packet. The firewall becomes a bottleneck.

The firewall also is single point of failure: If the firewall goes down, the whole network is affected. A firewall that goes down with no connection to the outside world, may affect the employee and customers trying to access the corporate web pages. If the firewall goes down, and no traffic is checked; the network is unprotected. Also, if a cracker manages to crack the firewall, he may have the whole unprotected internal network available at the strokes of his fingertips.

[6]But still are the majorities of wireless networks unencrypted. The "WorldWide WarDrive", which covered 4 continents and no less than 88122 wireless networks, only 33% was using WEP encryption. Homepage: `http://www.worldwidewardrive.org/`

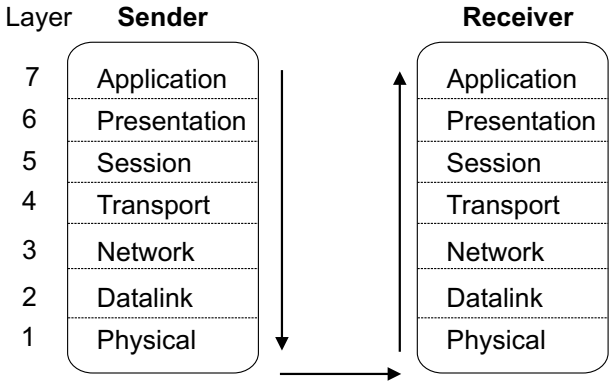

Figure 3.1: ISO's *Open System Interconnection (OSI)* reference model.

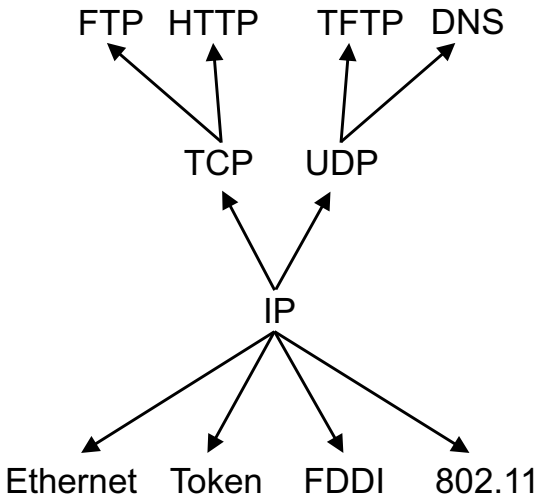

Figure 3.2: The "Internet" model. Also called the "TCPI/IP" model.

3.3 Types of firewalls

To understand how firewall works, some basic understanding of the underlying layers are required. All traffic that is transmitted across networks is broken into smaller packets and reassembled at the receiving host. Each packet consists of two parts; body and header. The body consist of the actual data, and the header is layer specific information. Two layered model exists; the "Open System Interconnection" (OSI) reference model and the Internet model.

ISO was one of the first organizations to formally define an architecture that described how to connect computers. Their architecture, called *Open System Internconnect* (OSI), is shown in figure 3.1. The OSI model is, today, not a protocol in itself, but rather a reference model. This is because another model became more popular.

At the same time ISO was hammering out the theoretical finesses of their model, Unix[7] shipped with a working network protocol using IP and TCP. Since Unix was popular and gaining foothold in

[7] Unix from University of California Berkeley. This Unix is the ancestor of the BSD family (FreeBSD, OpenBSD, NetBSD).

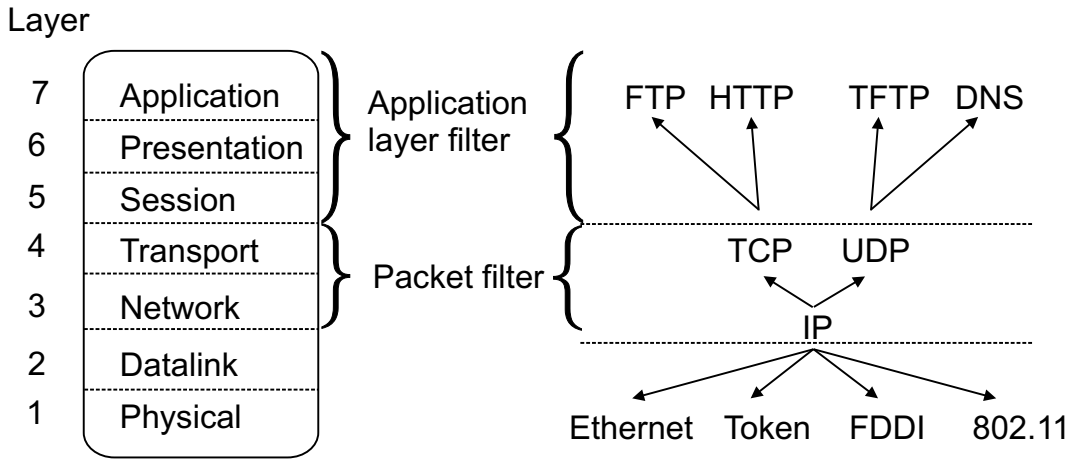

Figure 3.3: Application level firewall and packet filters placement in the OSI model.

academic and research institutions, TCP/IP became widespread. This is one of the reasons why any implementation of the OSI model, for example X.25, never took off.

The "Internet model" is also called the "TCP/IP model" after its two main protocols. The TCP/IP model is not only a reference model *per se* but a descriptive model as well. It has four layers, as see in figure 3.2 on the preceding page.

When an application needs to send some data over the network, it is pushed down the layers. At each layer, a layer-specific header is appended, before it is shipped over a physical link. It is clear that the TCP/IP model, with four layers are more effective than OSI's seven.

A firewall may filter traffic at different layers, see figure 3.3. The lower down in the model the traffic can be filtered, lesser resources are used: The faster the traffic is discarded, lesser resources is used to push traffic up through the layers. Some ethernet manufactures even provides some basic firewalls filtering at the data link layer. Read more about the two main protocols used in the Internet model in [32] (IP) and [33] (TCP).

Different firewalls exist to filter traffic at different levels. In some of the early classifications of firewalls, as Bellovin and Cheswicks paper "Network Firewalls" [6] and their famous firewall book [14], three types is defined: the packet filter, the application gateway and circuit-level gateway. The two latter are two different types of what today are called "proxies". So basically there are two types of firewalls: The packet filter and proxies.

3.3.1 Packet filter

A packet filtering firewall is probably the most used firewall on the Internet today: Its fast and widely available. The firewall consists of a set of rules (called a "chain") and the packet is sent into this chain for examination. If the packet passes all the rules, it is usually accepted and forward to the local process or network. If the packet does not pass the rules, it may be dropped or rejected. Usually there is more than one chain, as in Netfilter[8], there are three default chains (INPUT, FORWARD, OUTPUT) and new chains can be added by the user. A default action can be assigned to each chain

[8]Netfilter is Linux 2.4-2.6 packet filter subsystem. Read more in section 6.1.3 on page 46

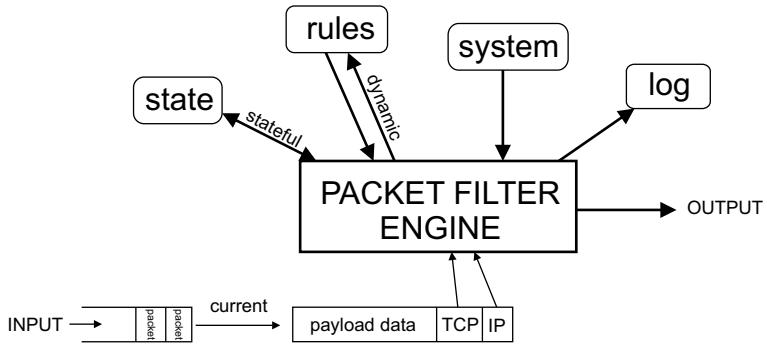

Figure 3.4: The packet filter model.

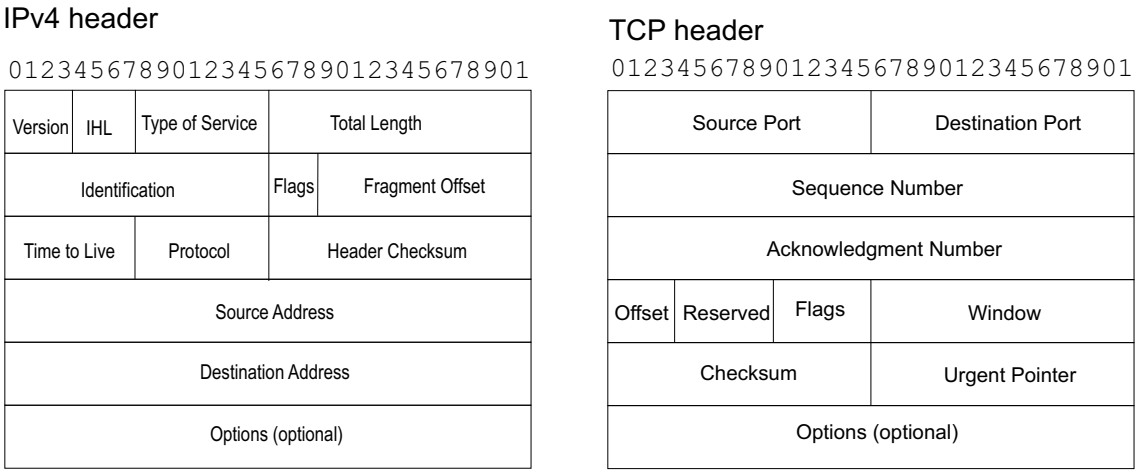

Figure 3.5: The IP and TCP header. The packet filter filters on various values in these headers.

(ACCEPT, DROP, ...).

According to Hallingstad et al. [30] a packet filter usually consists of (see figure 3.4)

- **The packet filter engine:** Is where that packets are checked against a rule-set.
- **Rule-set:** A set of filtering rules, based on some security policy and defined by a system administrator.
- **State table:** A table containing states of packets in flight. This can be packets that are part of a on-going stream to get a better understanding of the packets. For example, when a fragmented TCP packet arrives, it records a state, and forwards all fragments based on this "state". It is called a "stateful" packet filter. The first packet filters was "stateless" and filtered each packed individually.
- **Log mechanisms:** Keeping a log of the firewall is important.

Most packet filters does header inspection. The headers in question are the protocol headers: IP and TCP headers. Rules are built to filter on various fields in these headers, as seen in figure 3.5.

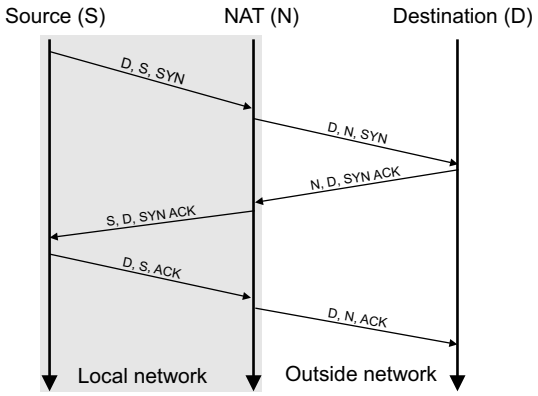

Figure 3.6: Network Address Translation (NAT)

The packet filter engine can filter packet based on several criteria; source/destination address, port number, TCP flags (SYN, ACK) etc.

The packet filter may also do filtering based on the body of the packet. For example; a new worm is unleashed on the Internet and is spreading fast. If the worm contains the sentence "Microsoft is evil", a rule may be added that drops all packets containing this sentence. Enabling body filtering poses a serious performance penalty; in contrast to header inspection, which have fixed field to inspect, the whole packet body must be inspected. It might only be partly successful if the worm is fragmented and the fragmented packet containing this sentence is not critical to the worms function. Since the packet filter don't have any understanding of what *type* of traffic it is examining, other legitimate traffic, such as a user request resources containing this sentence (web-page, text-document, email etc.).

The firewall used in this master thesis, uses a stateful packet filter. Read more under section 6.1.3 on page 46.

3.3.2 Proxy

A proxy acts as a "middleman" for all connections; all communication starts and ends at the proxy. When a host requests resources on an outside network, the proxy act on its behalf. When the connection is up, the proxy relays packet between the communicating parties.

A dedicated proxy has the ability to interpret the communication at the application level. This way the proxy have a larger understanding of the communication stream; A HTTP proxy understands the HTTP protocol and may deny ActiveX and JavaScript code. Since a packet filer operates on a lower layer, it has no understanding of the communication data. A dedicated proxy is also called application-level proxy.

There are also proxies that are caching proxies; hosts that keep a copy of all data they proxy. These proxies are designed for network efficiency and not security.

3.3.3 Network Address Translation (NAT)

NAT [41] was originally proposed as a short-term solution to the shortcoming of IPv4-adresses. NAT allows one set of (usually private) IP addresses/services to be mapped to a (usually a smaller set) of public IP addresses/services. This is especially convenient for small/home office (SOHO) or home users, who often have a dial-up/ADSL link and usually only get one public IP address. By using NAT, more than one computer can be connected to the Internet at the same time, by using the NAT'ed host as gateway.

The NAT'ed gateway is transparent to the inside users, and works by swapping the private source address (S) to, either a pool of public addresses, or one public address (N). The NAT keeps a table of all ongoing connection, and swaps the address back from the NAT'ed public address (N) and to the internal source (S). See figure 3.6 on the page before.

NAT has extended the lifetime of the IPv4 protocol, since the limited address-space may be squeezed even more[9]. It also destroys a fundamentally principle of the Internet; "end-to-end communication". It produces problems for a number of end-to-end protocols, like Mobile IPv4 [52] (roaming) and interfere with some encryption and authentication systems. With the arrival of IPv6, the need for NAT should no longer server a purpose.

Using the broad firewall definition found in [73], given at the start of this chapter, NAT *might* be defined as a firewall. NAT does hide the topology, and internal hosts can not be contacted directly unless some sort of port-based NAT (PAT) is used. But it is important to notice that NAT is not a firewall technology *per se*, as claimed in some literature including Hallingstad et al. [30]. NAT's goal was not security, but to extend the lifetime of IPv4.

3.4 Distributed firewall

Some claims, due to the limitations on the traditional firewall given, that the firewall has become obsolete. A new approach, that counters many of the limitation of the traditional firewall, was first described by Bellovin, in his paper "Distributed firewall" [5].

3.4.1 Network topology

In "Network Security Essentials" by Stallings [66], "Linux Firewalls" by Ziegler [71] and "Building Internet Firewalls" by Zwicky et al. [73] they all describes and outline a firewall as something that protect all traffic flowing through a single point (the gateway) between the inside and outside network. They operate with a clear distinction of "inside" and "outside" network topology.

This distinction does not always apply to the current and future network topology. When an employee is travelling, he might take his laptop, or other wireless gadget, and still be connected to the Internet. Still, he might need access to corporate resources like mail and corporate documents. The basis of a strict outside/inside topology does not apply; the mobile users are no longer "inside" the corporate network.

[9] IPv4's 32bits address-space is soon to be exhausted. With an increase in gadgets and devices using IP that are being connected, more addresses are needed. The next generation IP, IPv6, has a 128 bits address space which gives 2^{128} numbers of available addresses (or 340.282.366.920.938.463.463.374.607.431.768.211.456). In July 2004, IANA reported that IPv6 was added to its root DNS.

3.4.2 Distributed design

A distributed firewall is bringing the firewall to the end-hosts. By doing this, it is possible to bring the firewall into a mobile network topology and other shortcomings of the traditional firewall described above. By using a distributed firewall *the management of policy is still centralized, but the enforcement of the policy is distributed.*

Bellovin lists three components to implement a distributed firewall:

1. **Policy language:** A language that states what sort of connection are permitted and prohibited. In this master thesis, this is equivalent to packet filtering rules used in a packet filtering firewall (explained above in section 3.3.1 on page 27).
2. **System management:** A management tool that changes and enforces the security policy. This is the master and client framework explained in section 6.4 on page 55.
3. **Safe distribution:** A security mechanism that safely distributes the security policy. Bellovin suggest using IPSec, but other "similar" protocols may be used.

This master thesis designs and implements the system management tool, using the built-in firewall rules as policy language, and TLS (SSL) for safe distribution.

3.4.3 Discussion

Counter internal threats

One of the main reasons to deploy and use a distributed firewall, as I wrote in section 1.3 on page 2, was to counter internal threats. Since all the end-hosts, or at least the gateways on the internal network, are protected by a firewall, an internal attack is, if not countered, made difficult.

If distributed firewall is used alongside a network intrusion detection (NIDS), the attacker may be traced. If the distributed firewall was cooperating with a NIDS, firewall could counter and block the attacker, before he could do any damage.

Mobile networks

The once so clear, inside/outside topology is blurring. Mobile users may roam from one network to another and still request access to internal cooperate network resources. If a mobile node gets compromised, this can be a serious threat to the corporate network.

The other main reason for this master thesis was to counter this threat. A distributed firewall may still protect mobile users; the firewall policy is distributed securely and the mobile node may be protected.

Fine-grained control

Different protocols can't be filtered easily, such as UDP traffic. A traditional firewall may have a hard time determine whether this UDP-packet is part of an ongoing conversation or is new. The end-host,

however, knows. So by letting the end-host filter the traffic, a finer grained filtering is achieved and end-to-end communication may be restored.

A traditional firewall can not filter encrypted traffic. A distributed firewall may be able to filter this kind of traffic, since it can do the filtering after the traffic has been decrypted.

Simpler rule-set

A traditional firewall, located at the gateway, may have very strict default policies. If a user want to use a special service which require special privileges, he may be denied the opportunity, since the system administrator doesn't want to clobber up the rules in the firewall. If the system administrator is adding rules to satisfy the user, the added complexity of the rule-set increases the chance of introducing errors, as discussed in section 3.2.1 on page 23.

By using a distributed firewall, a host can have specialized firewall rules to met special requirements. The gateway firewall can have a simplified rule-set, which improves efficiency (the traffic doesn't need to pass through a huge rule-set). Since the firewall rules are simpler, there is smaller chance of configuration errors. But by using a distributed firewall, there is an increased *number* of rule-set, each which may contain (small set) of errors.

Hierarchical rules

To extend the distributed firewall; it may be configured to follow users and not hosts. This can be accomplished by enabling the client to "request" a users firewall rules from the management node (or a database under the management nodes control) when logging in.

Spreading the firewall to the end-hosts puts great care in crafting the firewall rules. One possible approach would be to create a hierarchical rule-set:

- **A global rule-set** common for all nodes in the network. These rules may cover the most basic firewall rules; ingress/egress filtering, blocking P2P traffic, blocking traffic from some known evil-sites etc.
- **Group specific rules** common for a group or division of a firm who usually runs the same set of software and uses the same set of services. For example: the computers in classroom X must be blocked for all IRC traffic and have restricted access to the teacher's file-server.
- **User/Host specific rules** are the last and most specific rule-set. A host known to create malicious traffic, for example a kiosk-computer available to all students, may have strict rule-set. The nodes user "Y" is using must also have a strict firewall rule-set, since he has done some wrongdoings before. On its most advanced level, the firewall rules follow the users. So despite which computer he logs into, the firewall rules follow him.

No single point of failure

To use a single traditional firewall to protect the internal network can be devastating if the firewall goes down. If the firewall shuts down, and all traffic is blocked, then potential customers and employees may not be able to do their work. If the firewall shuts down, and all traffic is allowed, the internal network is exposed. By using a distributed firewall, this problem is avoided: there is no

longer a single point of failure. If a firewall goes down, it only affects *one* hosts and not the whole network.

Gateway-firewall

Even if the distributed firewall spread the security policy onto the end-hosts, it does not mean that the traditional firewall is obsolete. If the network has a strict outside/inside topology, the traditional firewall may contain few global rules to block obvious attacks, do ingress/egress filtering etc. This removes the performance penalty dealing with huge rule-set and the possibility of misconfigurations.

Management tool

To manage a small set of firewall, which in part may have several dozen rules on different chains, can quickly become a maintenance nightmare. If one of them contains a configuration error, it may even become a source of false security. To make a firewall distributed, to spread out the rules on a large number of hosts, it puts great trust in the management tool. It must be easy to use and give good (graphical) overview of the network and be responsive. It should also be designed as to not shut itself out from a host (block all traffic from the management host).

Chapter 4

Intrusion Detection

> "If you torture the data enough, it will confess."
>
> — *Ronald Coase*

Intrusion detection is based on the assumption that it is impossible to avoid security breach in the long run. All defensive security mechanism may, at some point fail. Traditionally, a defensive approach was used to block the attacker. Intrusion detection is trying to identify, preferably in real time, attacks and assess damage caused. It is also based on the assumption that the *"exploitation of a system's vulnerabilities involve abnormal use, of the system; therefore, security violations could be detected from abnormal patterns of system usage"* [21]. How to measure and detect these "abnormal patterns" is not an easy straightforward task. Today, intrusion detection is gaining a lot of research attention.

4.1 Why?

Stallings [66] stress the importance of using some form of intrusion detection: *"Inevitably, the best intrusion prevention system will fail. A system's second line of defense is intrusion detection"*. At the same time, an intrusion detection grabs a good deal of resources: A powerful system being able to parse huge amount of network traffic, some intelligent logging facilities, and some human interaction is usually required. These are resources corporate firms may not have.

Dorothy Denning was one of the first to present an intrusion detection model in the classical IDS paper "Intrusion Detection Model" [21]. Lunt later redefined Denning's model and implemented the "intrusion detection expert system" (IDES) in [48]. Denning says the arguments for developing an intrusion detection is motivated by four factors:

1. Most system has security flaws, and fixing them all is not always an option due to technical or economical reasons.
2. A system that has flaws is not easily replaced with more secure systems, often due to economical reasons.
3. Developing totally secure systems is not possible.
4. Even the most secure network is vulnerable to abuse by insider who misuses their privileges.

4.2 The intruder

The worst kind of attacks is possible an intruder. Not only may he gain access to information he is not supposed to, he may do subtle changes that breaks integrity. In [3] penetration is defined as *"the ability to obtain unauthorized (undetected) access to files and program or the control state of a computer system."* According to [29], 39% of all attacks and misuse detection was system penetration. While 98% uses some kind of firewall, a steady 68% uses some form of intrusion detection (IDS). The commercial IDS has yet to gain foothold, as clearly seen in Norway, as only 20% uses some sort of IDS [74].

An intruder is most often an active attacker. All intruders gain more access to a system, using unauthorized methods (attacks). One of the first studies available on IDS, are a report published as early as 1980 called "Computer Security Threat Monitoring and Surveillance" [3]. In this report, Anderson classifies three types of intruders:

- **The masquerader:** Usually an outsider who penetrates the systems access controls to gain unauthorized access. The attacker using an ordinary user's account.
- **Legitimate user:** Called "misfeasor" in [66]. An insider who abuses his privileges to gain unauthorized access.
- **Clandestine user:** Can be either an insider or outsider who seizes control of the system to evade IDS and logging. This makes the clandestine intruder very hard to detect.

4.3 Damage control

A topic often neglected in intrusion detection is what to do *when* an intruder has been detected. Often sending some sort of alarm to a surveillance station is the only action taken. Then some human interaction, who usually must have a good understanding of computer networks and security, is needed to determine course of action. The alarmed user then does some sort of damage control: unplugging/shutting down the system, find the events in the logs, reinstall and remove malicious code.

According to the CSI/FBI [29] report, only 20% reported computer intrusion to law-enforcements. The corresponding Norwegian "Hidden Statistic Survey" [74] only 187 reported computer incidents was registered and of those was 76.3% concerning computer equipment theft. The fear of bad reputation is what usually keeps these incidents "in-house".

4.4 Different types of intrusion detection

There is traditionally two main classifications of intrusion detection systems. Porras [53] divide intrusion detection into two main types: statistical and rule-based. The other classification is between host based and network based intrusion detection.

4.4.1 Statistical anomaly detection

The statistical approach uses various statistical methods to extract metrics that model the behaviour of a user. Porras further split this category into *threshold* and *profile* based detection. Threshold

detection defines global threshold independent of users. Profile based detection tries to build a profile for normal behaviour for each users.

Anomalies detection has the advantages of detecting new attacks, since it is not based on pre-defined rules. The limitations are that statistical detection require huge amount of statistical data, often data that are attack free, to build the threshold/profiles. The most popular open source anomalies detection is Prelude (discussed in section 9.2.3 on page 90).

4.4.2 Rule-based detection

A rule based approach has a set of rules that defines normal behaviour. Porras further specified this into *anomalies* and *expert* systems.

A rule based anomalies system is similar to statistical anomaly detection. A huge amount of data is parsed and rules are created based on previous behaviour pattern. A lot of challenges must be met by rule based anomalies: What is to be monitored? How do initial rules for a new user get installed? Using some sort of "basis-rules"? A period of "learning-time"? But a learning time require an attack-free period. How could an attack free learning time be simulated?

A rule-based expert (penetration) system, also called signature based, uses pre-defined signatures to recognize attacks. Signature based detection is based on the assumption that all attacks leave their own unique signature that can be detected. The problem here is that it becomes a "race" between rule-writers and attackers; when a new attack is discovered, a new rule must be written, distributed and installed.

This master thesis outlines how to use a rule-based expert detection to detect network attacks.

4.4.3 Network and Host based

A *Host Based Intrusion Detection* deals with the "state" of the host. Usually it scans critical files for changes, or analyzes system call patterns as described in [31]. This master thesis implements and uses a host based intrusion detection called "PIDE", discussed in section 6.2 on page 47.

Network Intrusion Detection (NIDS) eavesdrop on the network traffic looking for suspicious behaviour. A popular open source rule-based expert system is SNORT[1]. SNORT has a wide range of rules available, and new ones are actively being produces as new threats arrives[2].

4.5 Problems and future direction

IDS is plagued with limitations. Since an intrusion detection must parse a huge amount of data, there will always be false positives (false alarms); incorrectly concluded that an intrusion occurred. This occurs if an IDS is configured to strict; a huge amount of false positives becomes generated that again may weaken the integrity of the IDS ("No worry, just another false alarm").

[1] SNORT's homepage can be found here: `http://www.snort.org/`

[2] Since SNORT is signature based, the "race" between signature-writers and new attacks has resulting in a "signature file" that is updated every 30 minutes found on the SNORT homepage. Retrieving and installing this new rule-set could be automated, but since it is retrieved over unencrypted HTTP and no MAC is included, it is vulnerable to man-in-the-middle attack.

If an IDS is to lax configured however, there can be too many false negatives: real intrusions not detected. A balance between a strict configured IDS giving few false positives is the goal. But as Templeton et al. concludes in [67] *"No current intrusion detection is 100% correct."*

In the recent years there has been an increase in research on intrusion detection. With the arrival of wireless nodes, which again may be running in ad-hoc modus with no centralized base-station, a cooperative intrusion detection may be used. But many challenges must be solved: Who is to be trusted? Should suspicious traffic and intrusion conclusion be shared among the nodes? What message exchange format should be used[3]? The wireless node often has limited computation and battery to parse and process huge amount of traffic. These, and many other, question are not answered here but are research topic for other works.

[3] A IETF "Intrusion Detection Exchange Format (idwg)" working group is working on this problem, and has currently submitted IETF drafts: `http://www.ietf.org/html.charters/idwg-charter.html`

Chapter 5

Scenarios

> "He never bore a grudge against anyone he wronged."
>
> — *Simone Signoret*

The distributed firwwall implemented in this master thesis, was intended for small to medium sized network, but may also be used on wireless ad-hoc networks.

5.1 Scenario 1: A small network

A small firm have a network consisting of <50 nodes, as seen on figure 5.1 on the following page. They are conducting its business over web. To handle support and payment, a database server holding the payment information, credit-card number and personal information, is stored in a database server. Sometimes the firm hires external consultants, when workload is high. The network has only one subnet.

The distributed firewall is present at the gateway, which protects the internal network from attack from the outside, by protecting traffic in and out from the gateway. The distributed firewall is also present at the web- and database server (marked as 'X').

In addition to distributed firewall, a network intrusion detection may be running at both the gateway and servers. When a network attack is in transit, NIDS will detect and report that to the management node. A host based intrusion detection is also present, which (hopefully) should be able to detect a clandestine attacker (successful attack not detected by the network intrusion detection). In other words, the host based intrusion detection should be able to detect compromised hosts.

For example, an external consultant (depicted as a "pirate" on figure 5.1), hired in to help with the high workload, wants to get hold of the payment database with all the credit-card numbers. The consultant launches an attack against the database server.

If the network only had a traditional firewall, a firewall located at the gateway only, the internal network would be unprotected and the attack would probably go undetected. Since a distributed firewall is running on the database server, the attacker would have a hard time getting past the firewall. The network intrusion would detect the attack, and report the incident back to the management node. Based on the severity of the attack, various actions may be taken. The consultant may be fired, and the incident reported to law enforcements. If the incident was less severe, the administrator may

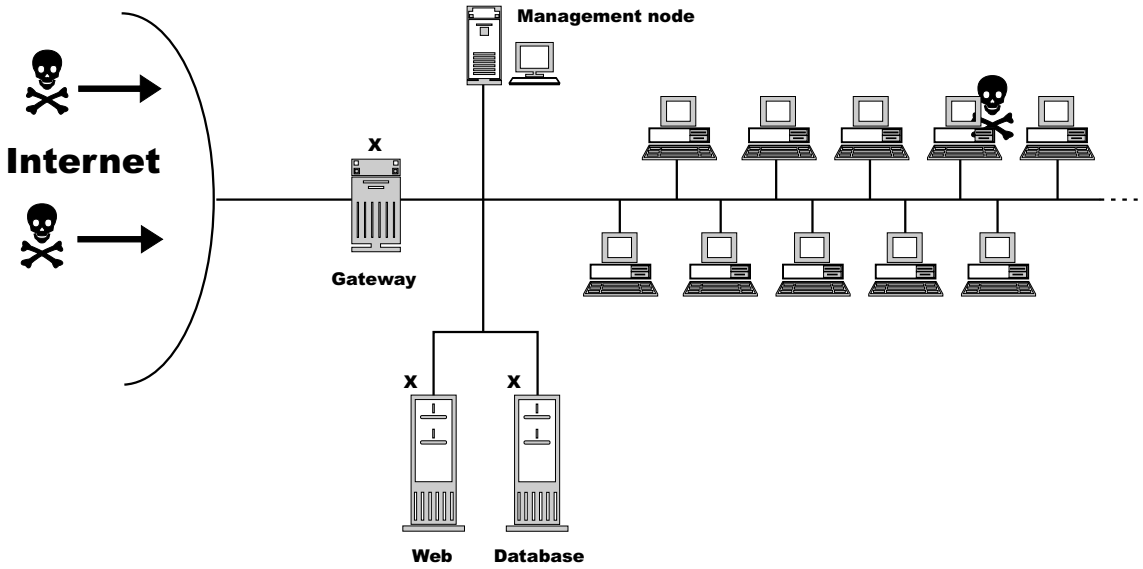

Figure 5.1: Scenario 1, a small network (<100 hosts)

craft some strict "user-rules" that prevent the user from launching such attack in the future. These user-rules follow the user, so no matter which host he tries to log into, the rules are enforced.

5.2 Scenario 2: A medium sized network

A system administrator has responsibility over a medium sized network (100-200 nodes). The servers are located on their own subnet (DMZ) and the nodes are located on several different subnets, see figure 5.2 on the next page.

This network may be a medium sized firm or a school. On this network, some users are using surfing on unwanted web-pages, some are using file-sharing (P2P) and some are spending all their time on IRC. Some or all of these activities, depending on the strictness of the security policy, are unwanted on the network; they steal network bandwidth[1] (P2P) and takes valuable time from working hours (IRC). These services also have a rather dubious security history (P2P and IRC is not known to "secure" a network).

By using a distributed firewall, the system administrator will have greater control over his own network. For example one of the subnets is a class-room in a school, where several of the students are spending time on IRC rather than paying attention to the teacher. The distributed firewall may be running on the gateway to each subnet (marked as 'X' on figure 5.2). By blocking all ICR traffic to and from the gateway, the IRC-clients will be crippling and students having no other choice than to pay attention to the class.

Another example being one employee (depicted as a "skull" on figure 5.1) which misbehave according to security policy. Different degrees of reaction may be executed based on the severity of the security breach:

[1]CacheLogic conducted a massive network survey on several large ISP from January to June 2004. It concluded that P2P stands for the double of ordinary web-traffic (10-times at peak) and that P2P traffic is increasing. The survey is found here: `http://www.cachelogic.com/research/index.php`

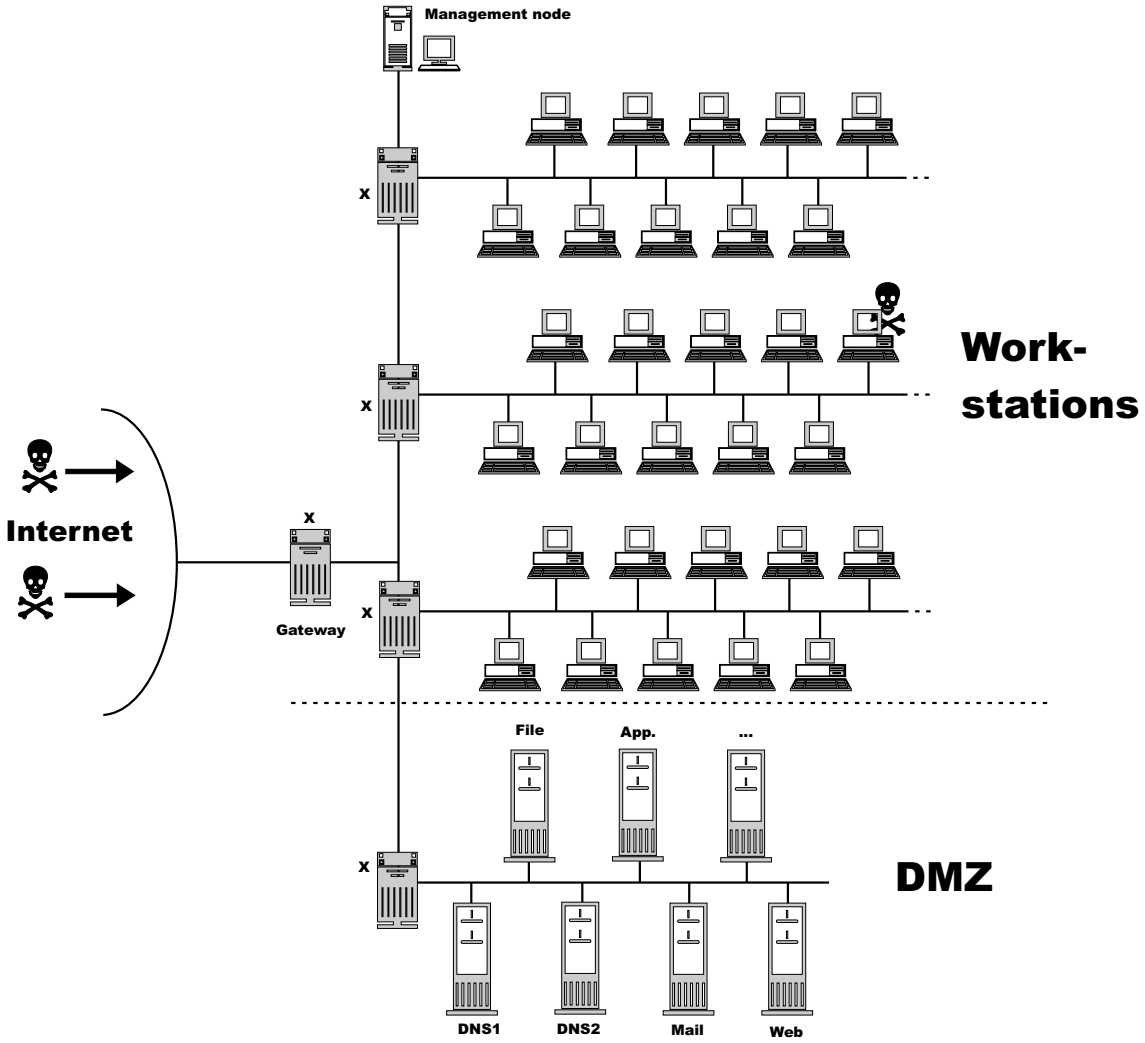

Figure 5.2: Scenario 2, a medium sized network (100-200 hosts)

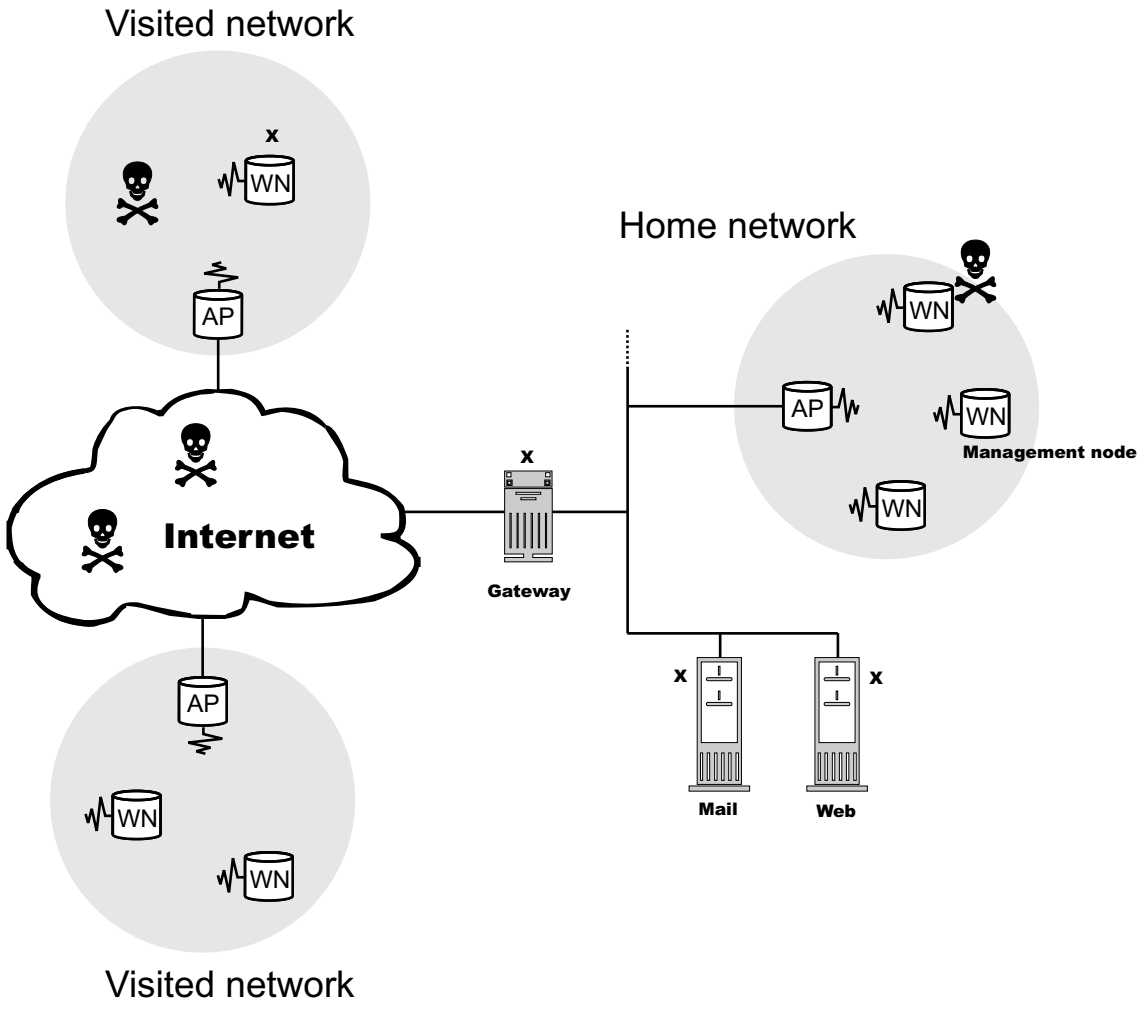

Figure 5.3: Scenario 3, a network with lots of mobile nodes

- If the employee is consuming to much network resources using P2P, that kind of traffic may be blocked in the gateway. Leave all other traffic open.
- If the employee is unusually creative to find non-work related network activities, more severe reaction may be enforced: All traffic, except traffic that the employee needs to do this work (for example surfing on the intranet and checking mail), are blocked.
- If the employee has done a serious violation of the security policy, or if the host has been compromised (perhaps acting as a zombie), the host can be denied all network access. If the distributed firewall is running on the gateway only, the host will only have access to its own subnet. If the distributed firewall is running on the hosts itself, all network traffic may be denied.

5.3 Scenario 3: A topology independent network

Today, there is an increase in wireless networks. Researching on next generation (wireless) routing is a hot topic. One of the latest new routing protocol for mobile ad-hoc networks is the “Optimized

Link State Protocol" (OLSR) [16][2]. The primarily purpose of this master thesis, as outline in 1.3, was not only to protect against internal attacks but also mobile nodes not behind a protective corporate firewall. A distributed firewall solves this problem.

For example, as seen in figure 5.3 on the facing page, all the employees have a laptop with a wireless connection instead of a (wired) desktop computer. The firm does a lot of consulting, and are often out to customers helping with technical problems or selling products. The firm also have a mail server and a web server. All employees must have access to both corporate mail and web server when travelling.

When a wireless node are leaving home, it is no longer protected by the gateway firewall. If a wireless node are visiting another wireless network, know to be insecure, a distributed firewall may protect the node; The wireless node 'WN', marked with an 'X' in the upper visiting network in figure 5.3.

When it comes to mobile hosts, a lot of enhancements may be added to the distributed firewall: A host can have two set of rules; one set of rules when the node is physically behind the corporate firewall, and another, stricter, set of rules when out on travel. Another feature may be when a consultant is out to a customer and need to install a program that uses some blocked ports, he may ask, either indirectly by sending a request to the management node, or directly by overriding the distributed firewall. These and other extensions are discussed in "Extension and further work" found in section 10.2 on page 93.

5.4 Scenario 4: A large network

On large networks, over 200 hosts, with different divisions managed by several system administrators, the distributed firewall implemented may still be used on parts of the network. Different system administrators, having responsibilities for different parts of the network, may use this tool on "their hosts".

To use the distributed firewall tool on a large network *may* work, but it is not designed for it. To extend the tool to support large installations, several enhancements as group management, graphical overview and user interface, more intelligent logging, are just some areas that must be worked on. I've discussed more thoroughly what needs to be done to support scalability in "Extension and further work" in section 10.2 on page 93.

[2]To read more about ad-hoc networks and how to use and set up OLSR under Linux, read my paper [45]

Chapter 6

Design and implementation

> "You can't solve social problems with software"
>
> — *Marcus Ranum*

The following chapter discuss the design and implementation of the distributed firewall. The tools used to implement is also discussed.

6.1 Implementation tools

A wide range of tools could be used to devlope the distributed firewall. Below follows a discussion of which devlopment tools, devlopment platform and firewall type should be used.

6.1.1 Development platform

GNU/Linux is a free operating system. Actually it is a lot more than just an operating system. Alongside there is often all the tools necessary to start development right away, either in C, C++, Perl, Python, PHP, COBRA, ...

In 1984, Richard Stallman [69], a hacker from MIT, was starting to get worried: The "hacking-culture", which embraced openness and sharing of academic ideas, was shrinking. A lot of fellow hacker was hired into new jobs and had to sign "Non-disclosure agreements" (NDA's). Some of the "openness" vanished. Ideas and source code could no longer flow uninterrupted, so Stallman quit his job and founded the "Free Software Foundation" (FSF)[1].

The FSF should extend and develop free software. They also had a goal of creating a Unix-like system which they would call GNU[2] (a recursive acronym for "GNU is Not Unix"). By the end of the 80's and early 90's, many of the most used tools was created; C compiler (GCC), Emacs editor, Gnu debugger (GDB) all licensed under the GNU General Public License (GPL)[3]. All source code licensed under GPL ensures that the code should be freely available, and every change/forking of the code must also be GPL licensed. Many other open source licenses exists[4].

[1] The "Free Software Foundation" homepage: `http://www.fsf.org`
[2] The GNU homepage: `http://www.gnu.org`
[3] The famous and controversy free software license GNU GPL: `http://www.gnu.org/copyleft/gpl.html`
[4] The "Open Source Initiative" (OSI) validates to check whether a license is open/free: `http://www.opensource.org/`

The GNU project lacked on major part, which was the kernel. GNU's own kernel project, HURD[5], had more or less stopped. This is where Linus Thorvalds and his Linux kernel came to the rescue. Together with the GNU tools and the Linux kernel, they became GNU/Linux which is gaining such widespread use today. Today there are literally hundreds of different Linux "distributions". The most common are RedHat, Suse, Slackware and Debian.

The distributed firewall implemented, may be used on any GNU/Linux distribution containing Python with kernel 2.4 or higher. It can also be used on any other Unix like operating system, containing Python and a scriptable firewall. It may even be run on Windows XP, which as of SP1 has support for a scriptable firewall.

6.1.2 Python

Python is an open source scripting language, initially created by Guido van Rossum. Since it is open source, it is freely available to download. Most GNU/Linux distributions already have Python pre-installed. Since Python is written in C, it is easily portable to other system (and so would this distributed firewall).

The Python language has a clear syntax, is object oriented (everything in Python is actually an object), and has numerous bindings to other language/libraries. When a lot of work needs to be done, for example implementing a distributed firewall, Perl code tends to become ugly pretty fast. Python is easier because of it strict syntax and object oriented nature. The Perl slogan "There is more than one way to do it" versus the opposite Python "There should be one– and preferably only one –obvious way to do it" says it all[6].

6.1.3 Firewall

While the Python code remain the same on all platforms, the use of firewall is not. The operating systems below all have support for a scripted firewall, but the syntax to do the same thing is different. By saying a "scriptable firewall", I mean a firewall that can be manipulated by using command line arguments. All the firewalls listed in the following operating systems are packet filters.

GNU/Linux

The first generation firewall in the Linux kernel (1.1) was a port of `ipfw` from BSD done by Alan Cox. Later Rusty Russel and Michael Neuling made significant changes to `ipfw` and, as of kernel 2.2, `ipchains` was released.

When kernel 2.4 was released, a new modular packet filter called Netfilter[7], written by Rusty Russel and other fellow hackers, was introduced. IPTables are the userspace command used to manipulate the Netfilter subsystem. A new API to manage Netfilter more easily, are rumoured to be underway; which could enable more tight integration with IDS.

[5]The HURD homepage: `http://www.gnu.org/software/hurd/hurd.html`

[6]The Perl slogan is often abbreviated to "TIMTOWTDI". The corresponding abbreviation of the Python slogan, "TSBOAPOOOWTDI", is not so often used.

[7]The Netfilter homepage: `http://www.netfilter.org/`

*BSD

The three main flavours of BSD Unix; FreeBSD, OpenBSD and NetBSD, all support a scriptable firewall.

NetBSD's firewall is IPFilter `ipf`. OpenBSD has of version 3.0 a new packet filter called PF using `pfctl` to administer. FreeBSD's firewall and traffic shaper program is called `ipfw`. Often can the various firewall implementations be used on other BSD flavours, both PF and IPFilter can be installed and used on FreeBSD as well.

Windows

Lately Microsoft has been starting to support more features by using the command line (maybe in response to the growing widespread of GNU/Linux?). As of Windows XP SP1, manipulating the Windows firewall by using the built-in command-line scripting utility called `netsh`.

6.2 Python Intrusion Detection Environment (PIDE)

PIDE is a host based intrusion detection. If a host is compromised, the attacker may take control of the distributed firewall daemon, and prevent the enforcement of firewall rules given from the management node. The attacker may even spoof the response to the management node, so it doesn't trigger any alarms. The compromised host may then be used to launch attack on other hosts. A compromised host *must* be detected. That is PIDE's job.

6.2.1 How does it work?

PIDE takes a "snapshot" of the system's state and later re-check that snapshot against the current-state. If the state has been altered, a report is generated. The "state" is an integrity database of key-attributes on selected files and directories on the host. Once this database is initialized it is used to verify the integrity of files/directories and discover new/deleted ones. When an attacker has compromised a host, it would most likely leave some traces behind (root-kit, changed files etc), that PIDE should be able to detect. But there is no guarantee; a skilled attacker may be able to hide his traces even from a PIDE.

6.2.2 Implementation issues

In my first paper outlining the design an implementation of the distributed firewall, the host base intrusion detection was supposed to be Tripwire or the Advanced Intrusion Detection Environment (AIDE). Tripwire had an "unfree" license for a long time, but is dual-licensed under GPL today. AIDE, which was GPL'ed from the start, was the open source community's response to Tripwire semi-free licence. Today both Tripwire[8] (last updated March 2001) and AIDE (last updated November 2003) seem to have no active development.

[8]The commercial version of Tripwire is active, but the Open Source version seems abandoned. `http://www.tripwire.com/`

I first started to write a wrapper for AIDE, but found it would be equally much work to get a functional wrapper as it would to implement Tripwire/AIDE myself: Sometimes AIDE produced errors that need to be caught, and I got no control over the report AIDE generated. Also, one primary goal was to keep dependencies of 3rd party software/modules down to a minimum. Neither AIDE nor Tripwire (the open source version) works under Windows. So I decided to write my own AIDE, which I called "Python Intrusion Detection Environment" (PIDE).

I reverse engineered AIDE by looking at the configuration file. At first I wanted at least compability between AIDE and PIDE configuration- and database file. Later on I dropped this requirement, since it would give a performance penalty: More (unnecessary) checks had to be included and the SHA-1/MD5 hash output had to be converted. I also added functionality that doesn't exists in AIDE, such as support for compressed and encrypted database and level of summary. PIDE has the following options:

```
> ./pide.py -h
usage: pide.py [OPTIONS]
pide.py (Python Intrusion Detection Environment) is an intrusion
detection system for checking the integrity of files. Basically a
python implementation of Tripwire/AIDE.

Mandatory arguments to long options are mandatory for short options too.
  -i, --init         Initialize the database.
  -C, --check        Checks the database for inconsistencies. This is
                     the default option.
  -u, --update       Checks the database and updates the database
                     non-interactively.
  -c, --config=FILE  Read config options from FILE
  -d, --default      View an example config file. Do a 'pide.py -d > pide.conf'
                     to pipe this to a config file.
  -e, --encrypt      Encrypt the database when doing init or check.
  -p, --password=PASS  The password to be used when encrypting or decrypting
                     database.
  -v, --verbose      Level of debug messages.
  -h, --help         Display this help and exit.

Report bugs to lars [at] gnist org
```

The configuration file pide.conf

Great care should be taken on fine tuning the PIDE's configuration file `pide.conf`. If not, a large number of false positives are generated. The configuration file is lengthy, and first takes some options regarding the database. The first setting is where the database and new database are located, whether it should be compressed and lastly if the database should be encrypted or not:

```
# Where is our database file?
database=file:/var/db/pide.db

# New databases (--init)
```

```
database_out=file:/var/db/pide.db.new

# Change this to 'no', or remove it to not gzip output
# (only useful on systems with few CPU cycles to spare)
gzip_dbout=yes

# Encrypt the database using AES with 256bit key?
# This require the blackbox module (and the Python
# Cryptographic Toolkit)
#encrypt_db=no

# Password to be used when encrypting/decrypting database
# NB! This should NOT be put here but given on the command line!
# You should have a *very* good reason for putting this here!
# For example, this file is not located at the host in question.
#password=secret
```

If the configuration file is badly tuned; for example checking files which changes normally, it can produce a lot of output. Or, if a host has been compromised, a large number of files may have been changed, deleted, replaced, log removed etc. A level of summary detail may therefore be set. There is also a setting whether to warn on dead symlinks or not:

```
# Length of summary when doing --check or --update
#   short  = a short report telling what's changed
#   normal = same as above and a list of changed files
#   detail = same as above and a detailed list of change
summary=normal

# Whether to warn about dead symlinks or not (default).
warn_dead_symlinks=no
```

An example of `short` summary produces something like this:

```
PIDE found differences between database and filesystem!!

Timestamp: Tue, 21 Sep 2004 13:24:55 +0000

Summary:
Total number of files in database: 1306
Total number of files on system  : 1307
Removed files                    : 0
New files                        : 1
Changed files                    : 1
```

The `normal` summary also lists which files it triggered on. Using the `detail` level, it also list what attribute it triggered on (timestamp, owner etc).

PIDE may also be run periodically as a cronjob, and certain environment variables may be exported when running PIDE. The `cron` daemon, which manages the conjobs, may read certain environment

p:	permission and file mode bits	a:	access timestamp
i:	inode number	m:	modification timestamp
n:	number of links	c:	inode creation timestamp
u:	user id of owner	S:	check for growing size
g:	group id of owner	md5:	MD5 signature
s:	size of file	sha1:	SHA-1 signature
b:	number of blocks		

Table 6.1: The attribute flags supported

variables. For example, if `cron` support to limit the number of lines reported from a cronjob, this can be set in the enviroment variable `LINES`. To define or undefine environment variables:

```
# You may specify environment variables to be exported when running here.
# It's normally only useful when used by the cron script.
# Define variable VAR to value val
# @@define VAR val
@@define MAILTO root@localhost
@@define LINES 1000

# You may undefine environment variables with
# @@undef VAR
```

Attributes

The rest of the config file contain a list of files and directories that will be scanned. The information of this scan will be saved in the database. The format is:

```
[!|=] entry [attributes-flags]
```

Where '!' signifies the entry is to be pruned from the list of files/directories to be scanned. The '=' signifies the entry to be added, but if it is a directory, then all the content are pruned (useful for /tmp).

The **entry** is an absolute pathname to a file or directory. The **attributes-flags** have the format:

```
[template] [ [+|-][pinug...] .. ]
```

Where '-' means ignore the following attributed and '+' means include the following attribute. The attributes supported are given in table 6.1.

To illustrate with an example: The following entry will scan all files in directory `/bin` and report any changes in mode bits, inode number, reference count, uid, gid, size, block count, modification and creation timestamp, and the signatures.

```
/bin +p+i+n+u+g+s+b+m+c+md5+sha1
```

R	[R]ead-only	(+p+i+n+u+g+s+b+m+md5+tiger+rmd160+sha1-a)
L	[L]og file	(+p+i+n+u+g-s-b-a-m-md5-tiger-rmd160-sha1)
N	ignore [N]othing	(+p+i+n+u+s+b+g+s+a+m+c+md5+tiger+rmd160+sha1)
E	ignore [E]verything	(-p-i-n-u-s-b-g-s-a-m-c-md5-tiger-rmd160-sha1)
> or Logs	growing logfile	(+p+u+g+i+n+S)
Binlib	binary files	(+p+i+n+u+g+s+b+m+c+md5+sha1)
ConfFiles	config files	(+p+i+n+u+g+s+b+m+c+md5+sha1)
Devices	device files /dev	(+p+i+n+u+g+s+b+c)
Databases	databases	(+p+n+u+g)
StaticDir	static directories	(+p+i+n+u+g)
ManPages	manual pages	(+p+i+n+u+g+s+b+m+c+md5+sha1)

Table 6.2: Pre-defined templates

The corresponding database will look like:

```
# This file was generated by PIDE, version 0.6
# Time of generation: Tue, 21 Sep 2004 15:13:38 +0000
# @@db_spec name perm inode size blocks lcount uid gid size atime mtime ctime md5 sha1
/bin/sleep 33261 175454 1 0 0 11132 24 0 996149092 1049561113 \
cbcfb2237f470e38952943e58f4fd1a4 0e03f2d9621b9caf5cb2330f5bd8367b6c48ea0f
/bin/zsh4 33261 176128 1 0 0 444684 880 0 1080239298 1082194179 \
bc11fd213b62f202a6e20a18130d56bb dc2595770070d34eefa60b33ade94e7de1ea1e04
/bin/lspci 33261 335069 1 0 0 24304 48 0 1080479892 1082194106 \
cce84e130715b463f8d907b363b0948b f98b6634b06526a91d5e0eb0152908ca3762713d
.......
```

To simplify the assignment of attributes to files and directories, they can be grouped into templates. A number of pre-defined templates have been predefined as shown in table 6.2

Templates may also be used with modifiers like:

```
/var/db/database.db        Databases+s
```

Customized templates can also be specified in the configuration file:

```
# Custom templates may go here
# NB!! '-' (minus) has precedence over '+' (plus), so the rule +p-p gives -p!
#MyOwnRule = +p+i+n+u+g+s+b+m+c+md5+sha1
```

Files/directories

The successfulness of PIDE is determined by its configuration file. If PIDE is monitoring files that changes a lot, like the users home files, a lot of false positives is generated. Below follows example entries for running PIDE under GNU/Linux:

```
# Homes
```

```
=/                  L      # First, root's traditional home
/root               R      # Most likely root's  home
!/root/.bash_history
=/home              L      # Holding the users homedir

# Log files
=/var/log           StaticDir
/var/log            Logs

# Kernel, system map etc. - files that are used by the boot loader.
/boot               Binlib

# System configuration files
/etc                R
/usr/local/etc      R

# Binaries
/bin                Binlib
/sbin               Binlib
/usr/bin            Binlib
/usr/sbin           Binlib
/usr/local/bin      Binlib
/usr/local/sbin     Binlib
/usr/games          Binlib

# Libraries
/lib                Binlib
/usr/lib            Binlib
/usr/local/lib      Binlib

# Databases
/var/db             Databases

# Test only the directory when dealing with /proc and /tmp
=/proc              StaticDir
=/tmp               StaticDir
```

For a complete working configuration file, consult the default configuration file `pide.conf` found in appendix C on page 129.

6.2.3 Requirements

There are two requirements when running PIDE:

1. The PIDE database must be initialized on a clean system. This is something that the system administrator typically does on a "fresh" installed system. *To generate a database on a compromised system will give false security!*

2. The database storing the "clean" state of the system must be inaccessible to a cracker. If the cracker breaks in, install a backdoor and at the same time update the PIDE database - PIDE will not report any break in!

There are two approaches to meet the second requirement; The first is to store the database on a remote hosts, for example the management node. The second approach, which is implemented, is to store the database encrypted. To encrypt the database, PIDE uses the `blackbox.py` module.

6.3 Blackbox - the cryptographic abstraction class

One of the main design goals was to keep the dependencies of third party modules down to a minimum. This sure posed a problem when dealing with encryption. The standard Python distribution does not include any cryptographic modules, only secure hash SHA-1 and MD5. Encryption is only available through third part modules.

6.3.1 Third party module

M2Crypto[9] is a crypto and SSL toolkit for Python. It depends on OpenSSL and SWIG, packets that may not be easily available and installed on all operating systems. Andrew M. Kuchling, a know Python developer, has created the "Python Cryptography Toolkit" [10]. It is a toolkit written from scratch with CPU intensitive algorithms written in C[11]. The toolkit supports a wide range of block-encryption algorithms and public-key algorithms. Since the toolkit does not have any third party dependencies and is supported on a wide range of operating systems (including Windows), this toolkit is used as the third party encryption module.

I quickly found out that to use this toolkit in an effective way, some sort of abstraction layer had to be used. On the toolkit's homepage, two such abstraction layers are mention, namely *ezPyCrypto* (which is a dead link) and *yawPyCrypto* which is an outdated project (not updated since Mars 2003). So I started writing my own abstraction class to ease the use of the crypto toolkit. The result became `blackbox`.

6.3.2 The need for padding

When doing encryption and decryption, different modes of operation may be used. Symmetric encryption has two different way of operate on plaintext: using block ciphers or stream ciphers. Stream ciphers operate on a stream of bytes and do not need any padding to nearest block. Block cipher operate on blocks of data, and must be padded to nearest block since most data don't divide on blocksize. Stream ciphers are (much) slower than block ciphers.

The use of a stream cipher would simplify the abstraction layers, since I wouldn't need to write a padding function. But when I did some tests, it soon became evident that stream ciphers are painfully slow. See the testing section 8.3 on page 82 for runtime data of this.

[9] M2Crypto homepage: `http://sandbox.rulemaker.net/ngps/m2`
[10] Python Cryptography Toolkit homepage: `http://www.amk.ca/python/code/crypto.html`
[11] Based on Schneier's bible "Applied Cryptography" [58].

The next problem was how to do the padding. The Schneier encryption "bible" [58] says *"Pad the last block with some regular pattern - zeroes, ones, alternating ones and zeroes - to make it a complete block"*. This is exactly what `blackbox` do.

6.3.3 The rest of the abstraction layer

The rest of the `reactor` class in blackbox contains functions to generate IV, encryption key, head generation and encrypt/decrypt.

Initialization Vector (IV)

The IV is generated by taking the SHA-1 of the current time and a random number. Since the IV needs to be at the same length as the block size, only the block size length is returned. If the block size is larger than the SHA-1 output (160bits), the hash is concatenated with itself until the correct length is reached.

Storing password

The encryption/decryption keys are generated by using PBKDF2, as described in section 7.3.2 on page 70, on a given password. When decrypting files, in this case PIDE database files, it should be some way of knowing when the password is wrong. If the password entered is wrong, a wrong decryption key is generated, which, when used, decrypts a database file containing nothing but garbage.

To implement a password check mechanism was a challenge: How does one store the password securely within the encrypted file? The password is hashed using the IV as salt using SHA-1. The SHA-1 produces 160bits (40 bytes), but to make it harder for an attacker only the 42 first bits (6 bytes) are stored. By doing this, the attacker will only be able to operate on a small portion of the hash, resulting in more matches when doing a brute-force attack. Of all these matches, only one of them are correct.

When this portion is checked against a password, the chance of another password generating the same (first 42 bits) hash is only 2^{-42}, which is more than acceptable.

Plaintext message format

Before the message is encrypted, the message is padded and certain "housekeeping" values are added: What kind of data is encrypted? Is the input data a string or some binary data? This is stored in the "code" field. The "size" field is the size of the plaintext (data) including padding. The plaintext is shown in figure 6.1 on the facing page.

Generate head

The head is applied to the ciphermessage. The head consist of part of the password as just described and the IV. The whole message is shown in figure 6.1 on the next page.

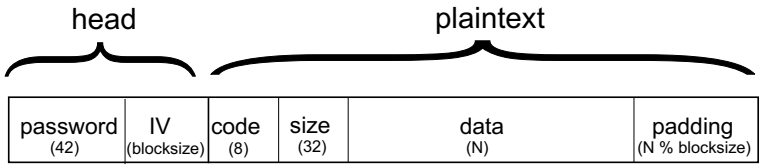

Figure 6.1: The message format

Encryption/decryption

The encryption first calls the PBKDF2 function which generates the encryption key based on password, described in section 7.3.2 on page 70. Then the message is formatted, as described in the previous section, and encrypted.

The decryption fetches the IV at the start of the message. Generates the decryption key based on password and decrypts the ciphertext. The padding is removed and the plaintext is returned. The password check-function is optional and may be called upon before calling decrypt.

6.4 Master and client

The `master.py` and `client.py` are the "main framework" of the distributed firewall[12]. The 'master' is the software running on the management station and the client is the software running on all the clients.

6.4.1 Master

The master should be running on the management node. The master has the ability to distribute security policies to clients under the master's control. The master framework contains several elements as shown in figure 6.2 on the next page.

The Shell

The shell is the system administrator's interface to the distributed firewall. The shell functions as an ordinary command interpreter; it takes command and executes them. The shell is modular in such a way that it can be replaced with any other interface (for example GUI) quite easily. When executing the master, the shell is provided instantly:

```
> ./master.py
Welcome to master ver 0.1 type 'help' for help
master>
```

The shell is now ready to accept commands. To enable "verbose mode", the following command may be issued:

[12]The source code for `master.py` and `client.py` are not included since they are pretty large. The newest updated code can be downloaded from: `http://www.gnist.org/~lars/studies/master/code/`

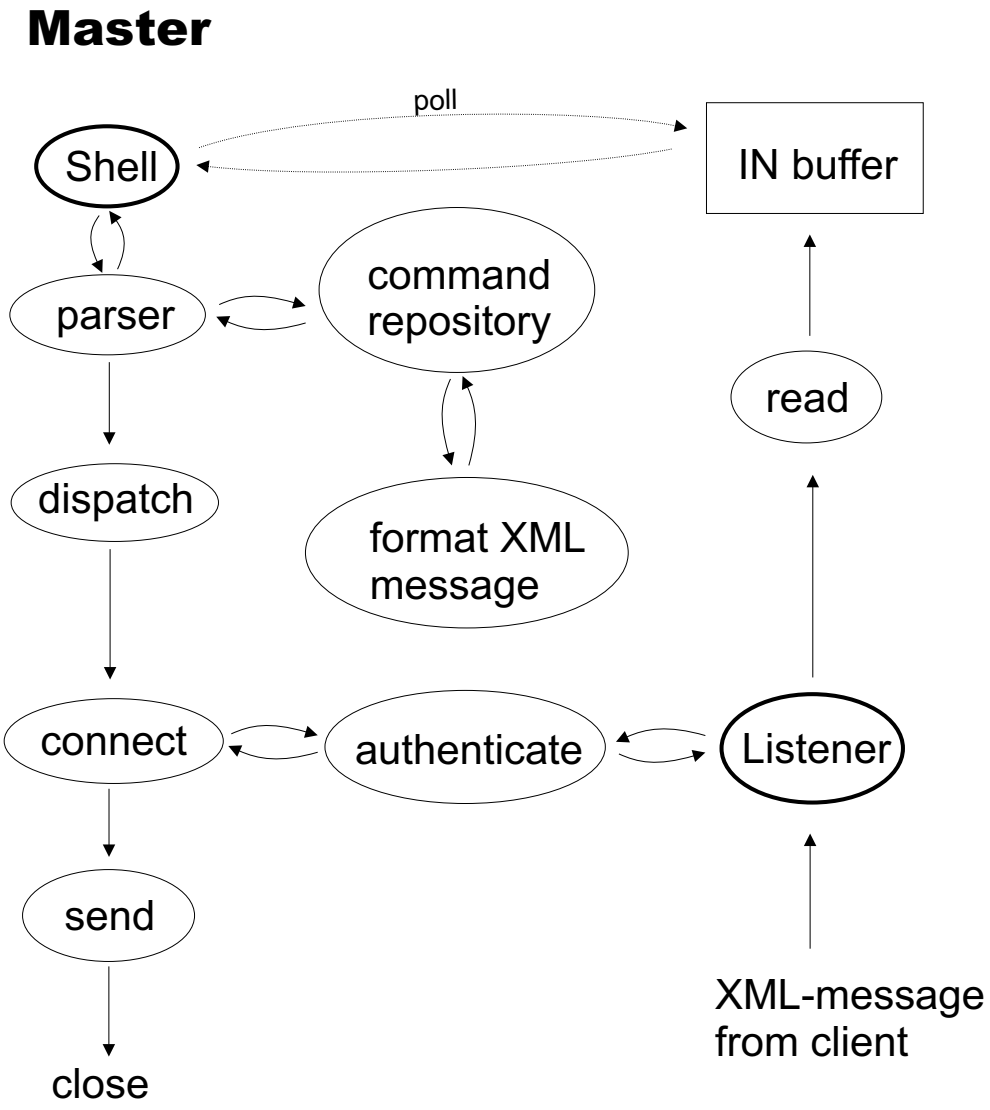

Figure 6.2: The MASTER framework.

```
master> set VERBOSE 1
Shell: parser: cmd finished!
master>
```

This modifies a global internal Python variable, which enables all verbose messages to be shown. This is usually only useful for debugging. As seen, a debug message appears right after the command: The `parser` function in the `Shell` class finished successfully.

Every commands typed is sent to the `parser` which validates the input.

Commands

The `parser` validates any given command and, if the command exist, sends it to the `command repository`. If the command does not exists, or is used incorrectly, an error message is printed. The command repository checks if all necessary files exists to complete the command; for example, if the command `ping` is given, then the command repository checks if the ping module is available. By issuing `help` in the shell, a list of all available command is listed:

```
master> help

*** master 0.1 ***

master is a command-line based application for distributed firewall
administration. A detailed information about use and the different
commands, can be viewed at: http://www.gnist.org/~lars/?action=studies

Below is a brief description on each of the commands:

  set              Modify built-in environment variables.
  help             Display help and exit
  fire             Change firewall rules of host
  ping             Ping a host
  pide             Python Intrusion Detection Environment
  poll             Any incoming client reports is polled from buffer
  exit             Exits the shell
  binexe           Execute command on remote host

Do a 'help <COMMAND>' for more extensive help.
```

Fire Has "subcommands" that can be executed:

```
master> help fire
Help for command 'fire':
'fire OPTIONS <node>'
Where options are one of the following:
    block N      - Block port N
    open  N      - Open port N
```

```
    flush        - Flush all firewall rules
    file <file>  - Run <file> firewall script
```

It is only the most basic firewall rules that are built-in. When issuing `block` or `open`, a hard-coded iptables command is executed on the remote host (in the INPUT chain):

```
os.system('/sbin/iptables -A INPUT -p tcp --dport %d -j DROP' % PORT)
```

These built-in commands are just examples, and more built-in commands can be added. All built-in commands must be wrappers to the `iptables` command that can potentially become a scripting mess: A huge amount of built-in wrappers must be added, and if something changes, it must be hard-coded. It gets even worse if support for additional operating systems not using iptables are added.

It is supposed that a "library" of network specified iptables scripts are created by the system administrator, that can easily be distributed though the `fire file <file>` command. So, for example, if the system administrator wants to block all bittorrent (P2P) traffic on host 'beinagrind', he may issue the command:

```
master> fire file torrent-block.sh beinagrind
```

Where `torrent-block.sh` is a shell script the system administrator has created that block all bittorrent traffic. All these firewall scripts should be found under the directory `firelib/`.

pide Has also a set of "subcommands" that can be executed:

```
master> help pide
Help for command 'pide':
'pide OPTIONS <node>'
Where options are one of the following:
    init   - Initialize the database.
    check  - Checks the database for inconsistencies (default).
    update - Checks the database and updates the database
             non-interactively.
```

The PIDE database is stored on the remote host, encrypted. The PIDE configuration file is stored on the management host under the directory `pidelib/`. So when a pide command is executed:

```
master> pide check beinagrind
```

A file called `beinagrind-pide.conf` must be found under `pidelib` and included in the message to the client.

binexe Is when a remote file is executed on the client. Everything after this command is tried executed on the remote host:

```
master> binexe beinagrind /bin/ls -lh /etc
```

Will try to execute `ls` with the options `-ls /etc` on the host beinagrind. This can be useful if the system administrator wants to, for example add a firewall rule directly using iptables, or check `dmesg` etc.

This command is likely to be removed in future versions. It is too much of a security risk to have it enabled: If the management node gets compromised, the attacker may send `rm -rf /` or `chmod 000 /` to effectively "kill" all clients. See also my discussion on a "command parser" in section "further work" 10.2 on page 93.

All the messages are formatted into an XML message, as described in section 7.1.2 on page 66. After the message has been properly formatted it is pushed down to the dispatcher.

Dispatcher

The dispatcher recieves a perfectly formatted message and handles the remote communication with the client. It connect to the client using secure SSL socket and then authenticate as described in section 7.4.2 on page 72.

After the message has been sent, the connection is closed. The management node does not wait for the client to finish the execution of the enforced command. If a node is in a wireless ad-hoc network[13], where links may be fluctuating, or if a node is roaming between two wireless access points or between manets, the connection may be broken and the command aborts. A command may take some time to complete (as it is with PIDE), and if the management closes the link as soon as the command has been sent, the risk of breaking the ongoing conversation is much "lesser".

Listener

After the client has executes (or tried to execute) a command, it connects to the master with any output and exit codes. The message received is in XML as described in section 7.1.2 on page 66.

The `Listener` is running in its own thread and is unaffected by the shell and dispatcher. As soon as a new connection is coming, a new thread is created to take care of the incoming request. The request is authenticated and, if successful, any content is stored in a global IN buffer. This global IN buffer is polled when the command `poll` is executed.

6.4.2 Client

The clients' responsibility is to take commands from the master node, execute the command, and send back a report on how that execution went. The client support several options:

```
./client.py -h
```

[13] Read more about MANET and its use in my Linux OLSR HOWTO: `http://www.tldp.org/HOWTO/OLSR-IPv6-HOWTO/`

```
usage: client.py [options]

options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         show debug messages
  -d, --daemon          start in daemon mode
  -lPORT, --listen=PORT
                        Listen to port PORT
  -cRCFILE, --config=RCFILE
                        use alternative config file, default: pyfire.conf
  -p, --printconf       View an example config file. Do a 'client.py -e >
                        pyfire.conf' to pipe this to a config file.
  -kPASS, --keypass=PASS
                        The shared secret between the client and management node.
                        Used in authentication.
  -nNRTHREADS, --nrthreads=NRTHREADS
                        Max number of threads to start
  -tMAX_WAIT, --time=MAX_WAIT
                        Max running time a program may have before we kill it
                        (default 1800)
```

If any of these options are used on the command line, they override any settings in the client's configuration file `pyfire.conf`. If a command given from the master "hangs", or takes too long time to finish, a default time before the client terminate the process may be given (the default is 30 minutes). An option to given maximum number of simultaneously active thread may also be given. This is to prevent a potential DoS attack to overwhelm the clients resources. If the client has more active threads, it denies all incoming requests until one threads finishes or is killed.

The client framework is simpler than the master, as seen on figure 6.3 on the facing page. It contains of three "main" part; the daemon (or listener), the executor and the pusher.

Daemon

When the client is started, it creates the Daemon class. The class is using the `SocketServer` module with the `ThreadingTCPServer` mix-in class to support threads (which again is just a mix-in class of `ThreadinMixIn` and `TCPServer`).

```
daemon = SocketServer.ThreadingTCPServer(('', PORT), Daemon)
```

The `daemon` creates a new thread as soon as a new connection is made. This connection is handled by the `BaseRequestHandler`, which is a mix-in class to `Daemon`:

```
class Daemon(SocketServer.BaseRequestHandler):
```

Then each connection is handled by this new thread. If the authentication is successful, the command is read and stripped for XML by the `parser`. The command is then sent, with the correct parameters, to the `command repository`. The `command repository` may save to file, if a file is included in the message (as it is with both `pide` and `fire file` commands).

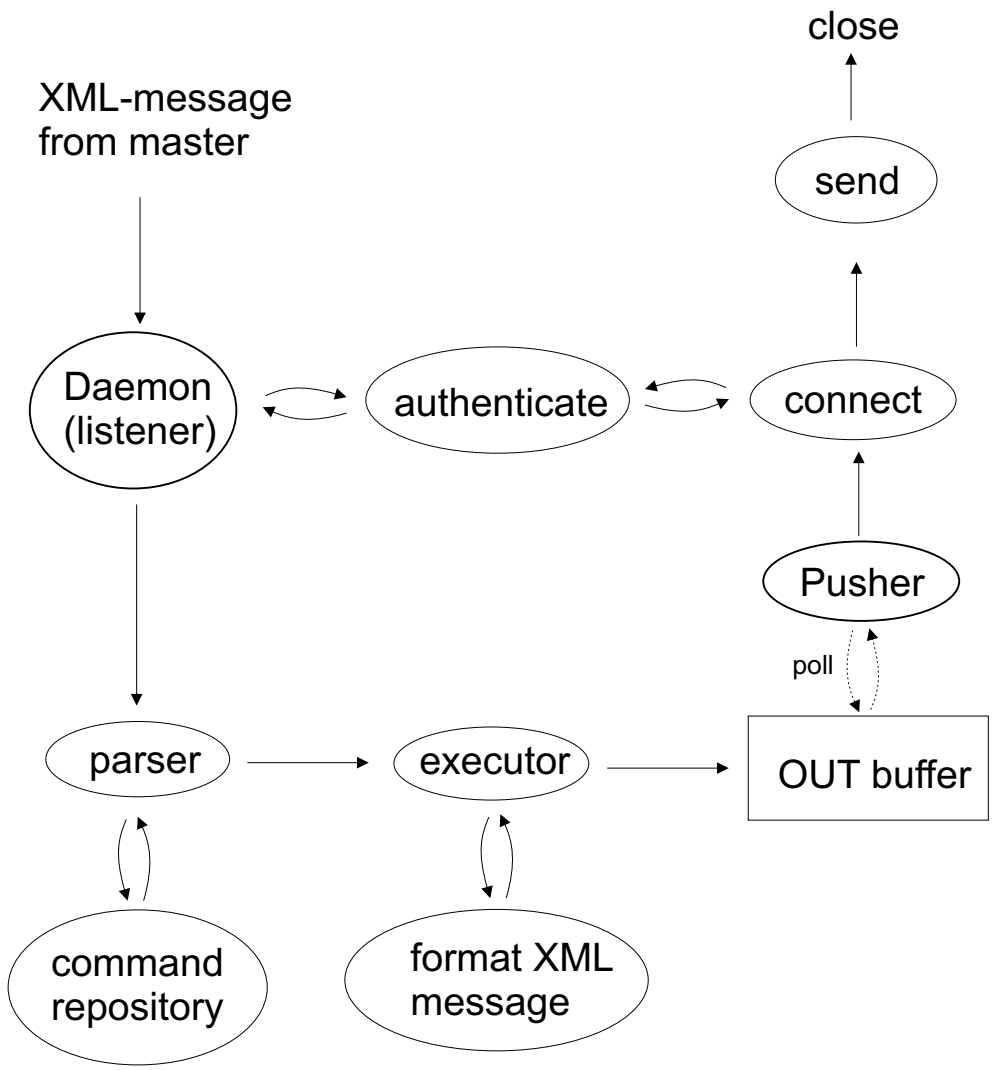

Figure 6.3: The CLIENT framework.

Executor

After the command repository is finished, the parser send the command to the `executor`. Executor tries to execute the command.

If it is some Python code (which none of the current command uses), it is executes inside the built-in Python function `eval()`. If it is a "remote execution", it tries to execute the given command located on the client. If the message contains a file to be executed, the file, which is saved by the `command repository`, is tried executed. Output from the command is captured and formatted to a XML message. Then it is pushed onto the OUT buffer.

Pusher

Alongside the `Daemon`, which spawns working threads, the `Pusher` is a separate thread. `Pusher` checks the global OUT buffer to see if it contains any reports to be sent back to the master.

If there is some content in the OUT buffer, the `Pusher` tries to connect to the master. If it is unsuccessful (the client is out of radio link), it tries again at a later time. Once connected and authenticated by the management node, the content of the OUT buffer is sent before it is flushed.

6.5 Auxiliary functions

The distributed firewall also has some auxiliary functions not directly related to manipulating firewall settings. These are functions that may help the system administrator to troubleshoot and obtain client information. Since these functions are not directly related to the distributed firewall, more function may be added later.

6.5.1 Ping

The command `ping` is used to create and send an ICMP packet. ICMP act as a higher level protocol, as it uses IP to send and receive messages, but RFC792 [38] specifies *"ICMP is actually an integral part of IP, and must be implemented by every IP module."*

Ping is perhaps the most used ICMP control message. It sends an ICMP echo message and the receiver sends an echo reply packet back to the sender (also called "ping" and "pong"). By waiting for a response, the sender can check whether or not the receiver is "alive". Not all types of ICMP messages should be allowed, as some can be used in active attacks. Blocking ICMP echo reply at the gateway is often considered a good thing; if not the attacker may just ping the whole internal network and gained a good understanding of the network topology. Internal hosts may ping each other. A real paranoid system administrator may block all ICMP traffic except to and from the management node.

First, I tried to write a python wrapper for the original ping that come with most operating systems. This way I could just do an `os.system('ping')` in python and capture the output. The problem quickly arouse when I wanted added functionality and found out that the command `ping` was not identical on all operating systems (different options to do the same thing, slightly different output that had to be captured). This quickly made the wrapper script large to catch all exceptions and make

a uniform behaviour on GNU/Linux, *BSD and Windows. When I also found out that different version of ping exists on GNU/Linux; I decided to write my own ping in Python. This way a consistent ping could be provided cross-platform.

The implemented Python ping can be view in section A on page 99. It is designed so it can be run as a stand-alone program, and used as an ordinary ping command supporting the most used functions:

```
> ./ping.py -h
usage: ping.py [OPTIONS] HOST
Send ICMP ECHO_REQUEST packets to network hosts.

Mandatory arguments to long options are mandatory for short options too.
  -c, --count=N    Stop after sending (and receiving) 'N' ECHO_RESPONSE
                   packets.
  -s, --size=S     Specify the number of data bytes to be sent. The default
                   are 56, which translate into 64 ICMP data bytes when
                   combined with the 8 bytes of ICMP header data.
  -f, --flood      Flood ping. Outputs packets as fast as they come back. Use
                   with caution!
  -6, --ipv6       Ping using IPv6.
  -t, --timeout=s  Specify a timeout, in seconds, before a ping packet is
                   considered 'lost'.
  -h, --help       Display this help and exit

Report bugs to lars [at] gnist org
```

It can also be used as a Python module. The management tool `master.py` as described in section 6.4.1 on page 55 uses ping as a module to ping hosts.

Since ping uses neither TCP (SOCK_STREAM) nor UDP (SOCK_DGRAM), the socket type SOCK_RAW must be used. This type, also called "raw-IP", makes it possible to construct the ICMP packet. The first challenge here was how to construct the packet. Since Python treats everything as objects, it made it hard to operate on a lower level required to manipulate bits in the ICMP header. Fortunate, Python has a module called `struct` that enables strings to be packet as binary data.

To create the ICMP header, it would be packed into a struct like this:

```
header = struct.pack('bbHHh', ICMP_TYPE, ICMP_CODE, checksum, ICMP_ID, ICMP_SEQ_NR+id)
```

Where the first argument to `struct.pack` is the binary C type. Here 'b' is a signed char, 'H' is an unsigned short and 'h' is short. Then the corresponding Python values are given.

The next challenge was to create the ICMP checksum. The RFC specifies how to generate the checksum using "The 16 bit one's complement of the one's complement sum of all 16 bit words in the header." Example of how to implement this checksum is given in RFC1071 [10].

There is however one big drawback of using raw IP. Only the superuser root is permitted to use SOCK_RAW. The only reason for ordinary users to use ping, is because it is setuid to root. So when users run ping, it is executes with root permissions. As a security precaution, no modern Unix kernels permit scripts to be setuid to root. So there is no help in setuid `ping.py` to root, it will not be run with roots permission. The only solution to make ordinary users use `ping.py` is to setuid the interpreter

python to root. But only a fool would do that. This setuid restriction was something I learned after the script was created.

Chapter 7

Implementation issues

> "Perfect technical security, like perfect physical security, is not possible."
>
> — *X.800, page 35*

During the implementation of the distributed firewall, I've encountered several implementation issues. Some have proven more challenging to overcome than others.

7.1 Discussion of message format

The message format sent between the management node and client nodes have evolved from a home-brewed format to XML.

7.1.1 Home-brewed format

The management station sends a command to the client, which executes (enforces) the command. After execution, the client sends a status message back to the management station, reporting whether the command was successfully executed or not. The commands from the management node may be python code, remote execution of command or a shell/binary file to be executed.

I tried several different message formats, before I've decided to use message with the fields separated by '%'. The originally message was on the form: <*size*>%<*type*>%<*name*>%<*options*>%<*data*>

So to send over a shell script, called '`block.sh`' with the options '`-tcp 90`', the message would look like this:

```
0034%2%block.sh%-tcp 90%<The rest of the shellcode, line for line>
```

This message format is not particularly flexible; each message fields are at a fixed position separated by a '%'. The first data that is sent, is the message size. This is to indicate how much data to read from the socket. The client first reads the size from the socket and then it reads size-bytes of data from that socket, which should be the rest of the message. Reading the size first, to determine how much data to read from the socket, is used when using XML messages as well.

To extend or change the message format, would require substantial changes to the message parsing function. What if the filename or option contains a '%'-symbol? The parser would think that this was a separator and the parsing would fail. This could have been fixed by converting the '%' to, for example the corresponding html code: '&#37'. But what if this code was to be present in a filename or option? The parser would misinterpret that code for a % and the conversion would fail again.

The solution to all these problems became XML.

7.1.2 XML

Extensible Markup Language (XML) has several advantages over other formats. One of XML's primary strength is application and platform independence; a XML message can be parsed and read by any XML ready application on any platform. This is perhaps especially important regarding government and library electronic documents that must be available many years from now; a closed source format may render the document unreadable in the years to come.

XML is rapidly becoming a standard when dealing with message format and data storage. From ordinary small configuration files or document file format in OpenOffice and Microsoft Office, to more complex protocols messages as in the new "The Intrusion Detection Exchange Protocol (IDXP)" using "The Intrusion Detection Message Exchange Format" both IETF draft as of writing[1]. There is even an XML Remote Procedure Call (XMLRPC) that has the ability to fetch XML data over HTTP making distributed message exchange even easier.

It is important to understand that XML is not a markup language, but a meta-language. The document structure can be created as long as certain syntax rules are followed. Different "sets" or types of XML documents, for example Docbook XML[2], are called a *schema*. A schema is defined (valid tags, nested tags etc.) by it Document Type Definition (DTD).

A XML document is said to be *well-formed* if it follow the general XML syntax rules, but may only be *valid* if it also follows the DTD.

The message format sent between the nodes in this master thesis uses are *well-defined*, but not *valid*. This is because a DTD is not defined or created. I found that using a DTD was not required to parse the XML message, as long as care was taken when creating the XML message. Future extensions to the XML message format may also include a DTD.

Management nodes command XML message

The message fields no longer have fixed fields in the message, but located between *markup tags*. This allowed for greater flexibility of what the content of each message-item (or *character data* as the XML people would call it). The message sent to from the management node to the client node:

```
<?xml version="1.0">

<ClientMessage>

    <node>DATA</node>
```

[1] Intrusion Detection Exchange Format (idwg) Charter: `http://www.ietf.org/html.charters/idwg-charter.html`
[2] Dookbook XML is an XML version of SGML and a popular document format.

```
    <type>DATA</type>
    <name>DATA</name>
    <options>DATA</options>
    <content>DATA</content>

</ClientMessage>
```

The first line is the XML declaration. This must always be present and states what XML version is used[3]. The declaration may also contain what character encoding used in the message (for example: `'encoding="UTF-8"'` for documents using UTF-8 as covered in RFC 2279). Since Python handles character encoding automatically, no character encoding is needed.

The following tags are used:

- **<node>:** Node defines what host (using hostname or IP-address) this message is intended for. This tag is not strictly necessary since the authentication process should make sure the message is given to the right host.
- **<type>:** The "type" tag defines what short of data is carried within the `<content>` tag. Four types of data types are defined:
    1. The data is some Python code that needs to be executed. This is done inside an `eval()` statement.
    2. The data is just a filename: The management node wants to execute a command remotely on the host defined in `<node>`.
    3. The data is a script file; usually shell code to execute iptables (firewall) commands. The content is base64 encoded before embedded (if the script file contains XML tags, the parser may be confused).
    4. The data is a binary file that needs to be executed. When a binary file is sent inside a XML message, it is base64 encoded.
- **<name>:** The name of the embedded file or remote command to be executed.
- **<options>:** Any options to the command being executed.
- **<content>:** This is the content of the command, if a file is included. The content here is base64 encoded using Pythons `base64` module

Client nodes respond XML message

After the client has carefully executed the message (and thereby enforcing the security policy), a return message is sent back to the management node:

```
<?xml version="1.0">

<ClientResponse>
```

[3]The World Wide Web Consortium (W3C) derived XML from SGML. The current XML version is 1.0: `http://www.w3.org/TR/REC-xml/`

```
    <Name>DATA</Name>
    <PID>DATA</PID>
    <ExitStatus>DATA</ExitStatus>
    <Time>DATA</Time>
    <Output>DATA</Output>

</ClientResponse>
```

The tages defined are:

- **<Name>:** The name of the command, with options, executed.
- **<PID>:** This is the process identificator (PID) of the given command. If the process spawn multiple processes, the mother-process-pid is given.
- **<ExitStatus>:** The exitstatus of the given command. A successful completion of the command gives exit status "0". Any error gives another number, and the data from the output tag must be carefully examined to see what went wrong.
- **<Time>:** The running time of the command.
- **<Output>:** Any output given from the command. Both stdin and sterr are given.

One the management node, this information is immediately made available to the administrator.

Simple API for XML (SAX)

The implementation tool, Python, has several of the most important modules for handling XML messages: There are basically two main APIs to handle XML: by using SAX or DOM. Using SAX or DOM with Python is extensively covered in [40].

SAX is an event driven, stream based API first developed by the Java community. Before SAX, every XML parser implemented it own API, so application was built to use specific parsers. SAX changed all that, by providing an extra abstraction layer for applications to use.

By calling SAX "event driven" is that it uses callback when certain events (tags) occur. For example, when a start tag occur, a flags is set that enables another function to parse the character data. The flag is disabled when an end tag occur. This makes SAX fast, has small memory footprint, and it may operate in real-time. This makes SAX ideal for parsing (large) XML documents on hosts that have fewer resources.

Document Object Model (DOM)

DOM is the opposite of SAX; it loads the whole XML document into memory at once. One the whole XML three has been loaded into memory, data may be extracted/changed. Since SAX is stream based, complex XML documents requiring lots of flags to be set, are easier to be parsed using DOM. For example, extracting character data from a deeply nested tag, can be fetched by using a single call.

This master thesis uses DOM to parse the XML messages. This is because the XML messages used are small, which does not make the SAX any faster or using significant more memory. It is also

easier to extract data using DOM. Python also have a module called `minidom`. This module is a "light-weight" implementation of DOM and uses less memory than the fully fledged DOM module. *"Overall, `minidom` may be best for loading simple (not necessarily small) configuration files [...] and using it anywhere a 'little' bit of XML is needed"* [40].

7.2 Intrusion detection methods

As described in chapter 4, there are basically two types of intrusion detection; host based and network based. A host based intrusion detection operates within the scope of the host only. It may detect that a user is about to misbehave while logged into the host, for example if a user is initiating a local DoS attack to take the host down. Host based intrusion detection may also detected if a host has been compromised. If a host has been compromised, we can not thrust that the security policy distributed to that host is being enforced.

An example being an employee has been on a business trip and been connected to some unsecure network full of crackers. Or the kids of the employee has "played" with the computer and installed lots of bogus software which some guys on IRC claimed to "speed up the Internet connection". When a compromised laptop returns back to the firm's network and gets connected, it basically functions a backdoor into the firm's network. A host based intrusion detection should be able to detect a compromised host.

7.2.1 Network intrusion detection

Upon taking a special course in intrusion detection[4], I found out that implementing a network intrusion detection that would manipulate the firewall rules automatically would be to much work. I though about using SNORT and some agent to parse SNORT log files in real-time: Three steps are necessary to complete implement this:

1. Some well-defined SNORT rules that produces well-defined logs.
2. An agent that parses the snort logs in real time, and apply firewall rules based on hard coded snort to iptables relations.
3. The agent also logs any intrusion reports (severe enough) to the management station.

Not only does this require a third party program, SNORT, to be installed, configured and running - the most known attacks must be analyzed to generate "hard-coded" iptables commands. As soon as a SNORT logs occur, it is parsed and iptables rules applied. Later version of the distributed firewall may include support for network intrusion detection.

7.2.2 Host based intrusion detection

A host based intrusion detection is implemented. Python Intrusion Detection Environment (PIDE) records the state of the host in a "clean" state. Later that the current state is checked up against the recorded state. The design, function and implementation are fully discussed in section 6.2 on page 47.

[4]Read more here: `http://www.gnist.org/~lars/?action=ids`

7.3 Discussion of encryption keys

As already discussed in section 6.2.3 on page 52, two requirements must be met to enable PIDE work correctly: The initial "state" must be recorded at a "clean" state and the database containing this state must unavailable to an attacker. The second prerequisite can either be met by storing the database on a remote host, or encrypt the database.

PIDE prevents tampering of the database by encrypting it. The problem then quickly arouse: How do I create the encryption keys?

7.3.1 Hash output

My first solution was to take a SHA-1 hash of a given password and a random salt. SHA-1 produces a hash of 160 bits; if the key length required was shorter than 160 bit, it was truncated. Was the key length larger that 160 bits, it was just concatenated and truncated to key-length. The salt and a MD5 hash of the password was stored within the file. When the file was to be decrypted, a given password could be checked against the MD5 hash to see if it was the correct one. If it was, a SHA-1 hash of the password was used to produce the key.

This solution was a little to "simple" and "home-cooked" to provide a robust solution: Even if salt is used, the solution is much more vulnerable to brute force attack than "PBKDF2". A more robust solution to create encryption key from password is called PBKDFv2 or just PBKDF2.

7.3.2 PBKDF2

The "Password Based Key Derivation Function version 2" (PBKDF2) is specified in [43] from RSA Laboratories. It uses a pseudorandom function to derive keys. The functionality of PBKDF2 is implemented in `blackbox.py` (`class PBKDFv2`), see Appendix D on page 133.

RFC 2898 [43], specifies the following required steps:

1. The maximum length *dkLen* of the derived key *DK* is given as:

   $$dkLen > (2^{32} - 1) * hLen$$

   Where *hLen* denotes the length in octets of the pseudorandom function output, which is 16 for MD5 and 20 for SHA-1. If *dkLen* exceeds number, the function outputs "derived key too long" and stops.

2. l is the number of *hLen*-octet blocks in the derived key, rounding up, and r the number of octets in the last block:

   $$l = \left\lceil \frac{dklen}{hLen} \right\rceil$$

   $$r = dkLen - (l - 1) * hLen$$

3. For each block (l or r) of the derived key, the function F with input; password P, salt s, the iteration count c and the block index to compute the block:

   $$T_1 = F(P, S, c, 1)$$

$$T_2 = F(P, S, c, 2)$$

...

$$T_l = F(P, S, c, l)$$

Were function F is defined as the exclusive-or sum of the first c iterates of the underlying pseudorandom function PRF (using HMAC-SHA1 [44]) applied to the password P and the concatenation of the salt S and the block index i:

$$F(P, S, c, i) = U_1 \oplus U_2 \oplus ... \oplus U_c$$

Where:

$$U_1 = PRF(P, S \parallel INT(i))$$

$$U_2 = PRF(P, U_1)$$

...

$$U_c = PRF(P, U_{c-1})$$

Here, *INT(i)* is a four-octet encoding of the integer i, most significant octet first.

4. Then the blocks are concatenated and the first *dkLen* octets is extracted to produce a derived key DK.

$$DK = T_1 \parallel T_2 \parallel ... \parallel T_l < 0..r - 1 >$$

5. The derived key *DK* is returned.

Using PBKDF2 provides much better entropy with larger keys and is substantially harder to brute force. Read more about the use of encryption of files in "blackbox" in section 6.3 on page 53. This function is also used to generate keys during authentication, discussed next.

7.4 Discussion of authentication choice

The authentication mechanism is an essential part of the distributed firewall. Not only should it withstand "man-in-the-middle" attack, but also minimize vulnerability to DoS attack. No encryption or authentication services are provided in the standard Python distribution. *"Standard mix-in classes to implement various authentication and encryption schemes"* can be read under the "future work" section to the `SocketServer` Python module. Since Python has no support for encryption or authentication, I had to design my own authentication scheme.

7.4.1 Extensible Authentication Protocol (EAP)

Fortunate, there has been a lot of research on authentication. With the arrival of the new security standard 802.11i [35], an authentication method called 802.1X [34] is used. 802.1X in 802.11i uses the Extensible Authentication Protocol (EAP) [1] as authentication protocol. EAP has traditionally been used as an authentication protocol for dial-up. It is not an authentication protocol itself, but merely a protocol optimized for authentication. The real authentication is carried within EAP.

EAP supports a large number of EAP authentication methods[5]. In the dial-up days, a handshake using either PAP or CHAP [64] was used as authentication method. The most popular authentication used in wireless enviroments has been EAP-TLS [2]. EAP-TLS does a TLS [22] authentication both ways, requiring x509 certificates on both sides.

Today EAP-Tunneled TLS (EAP-TTLS) and Protected EAP (PEAP) (both IETF draft) are competing to become the *de-facto* authentication mechanism in 802.11i. They both set up an encrypted TLS tunnel in which another authentication mechanism may be used[6].

It would be preferred to use EAP-TLS authentication. Since the client needs to authenticate against the management node and the management node against the client. One of the arguments of using either EAP-TTLS or PEAP is that client certificates are optional. This simplifies configuration, as only certificates on the management node is required, but it only authenticate the management node to the client.

The distributed firewall need to authenticate clients against the management node and vice versa. EAP-TLS does an authentication both ways, and is the preferred authentication method to be used as the authentication method in the distributed firewall. EAP-TLS also creates keying material that can later be used for session encryption. Read more about 802.1X and key exchange in the paper I've written [46].

Unfortunate, implementing EAP with TLS creating EAP-TLS is a huge task[7], to huge task for a master-thesis. A simpler authentication method is therefore used; using SSL sockets with a handshake inside the SSL tunnel.

7.4.2 Using SSL-socket

Python has limited support for SSL. The `socket` module has support for starting a SSL session over an open socket, but the method does *not do any certificate verification*. Because of this serious drawback, another authentication method is used inside the SSL encrypted tunnel.

7.4.3 Authentication choice

So what authentication method should be used? Since there is an encrypted SSL tunnel, the authentication method should be safe from passive attacks like eavesdropping (read more about passive

[5]Each authentication must be using a unique EAP type to identify the authentication method used. The official EAP type registry is located at IANA: `http://www.iana.org/assignments/eap-numbers`

[6]The 802.1X authentication supported on Windows XP SP1 is PEAP with MS-CHAPv2 [72]. MS-CHAPv2 is a modified CHAP [64] challenge.

[7]Implementing EAP-TLS would require to implement three RFC, all which are large and complex: "RFC 2246: The TLS Protocol ver. 1.0" (80 pages), "RFC2716: PPP EAP TLS Authentication Protocol" (24 pages) and "RFC3748: Extensible Authentication Protocol (EAP)" (67 pages). Which are 171 pages of specification.

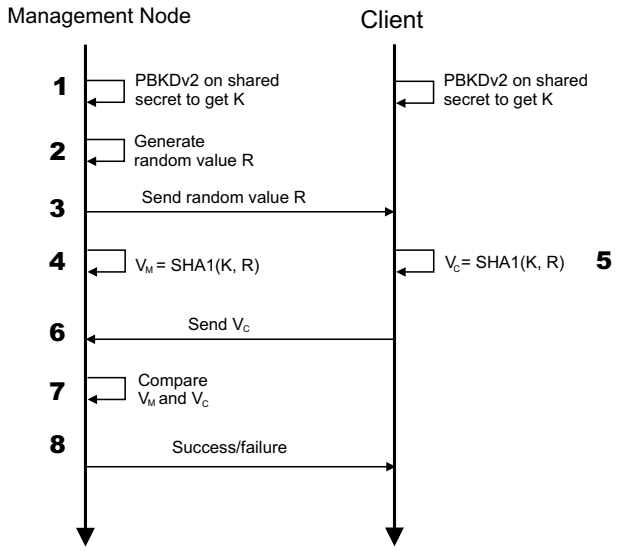

Figure 7.1: The three ways authenticate handshake.

attacks in section 2.2.2 on page 10). A three ways handshake (challenge response), as shown in figure 7.1, is used.

1. The client shares a secret with the Management Node. This shared secret is a password of at least 8 characters. Then the key derivation function "PBKDF2" as specified in [43] is used to generate the management and client key K.
2. The management node generates a random value R of 256 bits.
3. The management node sends the random value R to the client.
4. The management node computes V_M by using SHA-1 one way hash $sha1(K_M, R)$.
5. The client computes V_C by over the same values.
6. V_C is sent from the client to the Management Node.
7. The management node compares V_C and V_M.
8. If the values are identical, the authentication is successful, otherwise is fails.

The shared secret (the password) is not used directly. By using PBKDF2 a brute force become exceedingly hard since the entropy is increased and it is computative intensitive.

The client's shared secret can be given to the `-k` option or specified in the config file. The management node has a database file, `secrets.db`, with format:

```
# hostname 'shared secret'
beinagrind Secret77
konungblod getOff42
.....
```

Which has hostname/shared secret pair. When authentication occurs, the master looks up the shared secret, based on hostname. If the master can't find the shared secret, authentication fails.

7.4.4 Other solutions

Using public key encryption

Since the client and the management node uses x509 certificates (to initiate SSL), a two-way handshake could be used instead. The management node could generate a random secret and encrypt it with the client's public key. The client, which is the only one who can decrypt the message, could decrypt and re-encrypt the message with the management nodes public key. The management node could then decrypts and compares the message with the one originally sent. Only if the two messages matches, authentication is successful.

But since there are no Python modules for dealing with x509 certificates, this solution is not used. It could be done be using the third party program `openssl`, but would require a lot of "hacks" to make it work and would not become a robust solution.

Twisted Matrix Laboratories

Twisted[8] is a framework for using network protocols in Python. It has support for a wide range of protocols, including SSL and SSH. But it also supports IRC, HTTP, IMAP and other protocols. This makes Twisted into a large beast; the source code alone weights 10MB (which is one third of the whole Python source code). By using Twisted, not only is there a third party that may require security upgrade, but it is also a (to) large dependency. One of the design goals of this master thesis was to keep the dependency of third party modules down to a minimum.

7.5 Thread-based model

The simplest method for handling network traffic is synchronous I/O. It provides no concurrency at all; while waiting for the I/O operation to complete, the program stops. This method is easy to use and suitable for testing, but it is not usable for this implementation.

The management node and the clients need to handle concurrent events: The management node need to both take order from the system administrator as well as be listening for incoming reports from client hosts. When a command has been issued by the system administrator, it is assign it own thread for dispatching. This way, the system administrator doesn't have to wait for the command to complete connection and sending.

The client may also do several things at once; when a command is received from the management node, it is assign it own thread that handles parsing, execution and reporting back to the management node. If a command takes long time to execute (as running PIDE), another command from the management node may be handled concurrently by the client.

[8]Twisted Matrix Laboratories: `http://www.twistedmatrix.com/`

7.5.1 Asynchronous I/O

When dealing with multiple networked I/O, as both the client and the management node do, there are two options: using either multitasking or asynchronous I/O. Asynchronous I/O is a technique specifically designed to be efficient. All I/O requests are pushed down to kernel level and handled by the operating system in the "background". This reduces the number of context switches and resources used. This greatly improves performance if there is large amount of sockets and/or large amount of data sent over the network. It also makes it possible for one thread to handle multiple I/O requests at once. The `ping.py`, found in appendix A on page 99 uses asynchronous I/O.

The Python interpreter is not fully thread safe, so each thread has to acquire the Global Interpreter Lock (GIL)[9] before accessing Python objects. The GIL is released after 100 bytes of executed bytecode (interpreted Python code) or if the thread is waiting for I/O. Running threaded Python programs on multiprocessor machines therefore suffers from GIL restriction. This is why Twisted Python framework uses Asynchronous I/O.

Unfortunate using asynchronous I/O is more complex than using multitasking. It is recommended to use asynchronous I/O when dealing with large amount of data. Since the client/management node threads are going to do more than just sending and receiving network traffic, asynchronous I/O is not a suitable solution.

7.5.2 Threads

All modern operating system supports multitasking. Multitasking is to do more than one thing concurrently, either by using processes or threads. Using multiple processes to handle multitasking provides isolation of each process but no shared data. It is often accomplished by spawning multiple processes (also called "forking"). An example being the Apache web-server[10] which forks a new process for each incoming HTTP-request. There is also a security gain by forking processes; if one process crashes, the other processes are unaffected.

The uses of threads have several advantages over forking. Threads my share some internal data-structures (sharing data between processes are more difficult). Creating a thread takes fewer resources than forking a process[11].

When using threads, the thread can be interrupted at any time (to give execution time to other threads). If a thread is interrupted when updating some shared data-structures, data corruption may occur. So when accessing shared-data structures, a mutual exclusion object (mutex) is used. A mutex is a lock that enables multiple threads to access the same data-structure, but not at the same time. To access shared data, the lock must be acquired. The Python `thread` module have lock objects `allocate_lock`, `acquire` and `release`.

Both the client and management node must be able to do more than one task at the time. I'm therefore using the `threading` module for some tasks, and the more sophisticated `SocketServer` to generate the listening service on both sides. When a new connection is received, the SocketServer assigns the request to a thread.

[9] Read more about GIL in the official Python documentation: `http://www.python.org/doc/current/api/threads.html`

[10] As of Apache 2.X, threading is also supported.

[11] Traditionally forking processes are more optimized under Linux than Windows. While the Linux (and Unix) tradition has used forking, Windows rely more on the use of threading.

7.6 PKI or not to PKI

Public key cryptography, as described in section 2.3.1 on page 18 depends of the distribution of the public keys. The problem being that the keys need to be distributed in such a way that the key received really is from the one it claims to be. If Alice wants to get Bob's public key, and Bob email it to her, the mail could have been intercepted and replaced with the attacker's key (see section 2.2.2 on page 13 for more about "man-in-the-middle-attacks"). The distribution *method* needs to *authenticate* Bob to Alice, and provide a *digital signature* of Bob's key that Alice can verify. This can be done by public key cryptography, but there is no key to be trusted (a typical "chicken and egg" problem). This is where "Public Key Infrastructure" (PKI)[12] comes to the rescue.

PKI uses (x509) certificates that basically is a public key (with additional information), *signed* by X. X is here usually a *certification authority* (CA). The use of certificates does not solve the distribution problem directly. The CA's public key must be known and available to all potential participants[13]. If a user has and trust the public certificate to CA 'X', he can validate certificate from 'Y' if it is signed by 'X'. The idea is to build "chains of trust" or, a hierarchical tree structure with CA at the top ("root").

When trusting a root CA, it must be certain that the CA only issues certificates to subject which has proven their identity. If a CA is issuing certificates to subject *without* checking their identity, the whole "chain of trust" becomes meaningless. A PKI does not only consist of a root CA, but must also consist of a "Certificate Revocation List" (CRL) which hold a list of certificates which not is to be trusted.

The PKI framework is large and complex, and is primarily targeted at large installation or large user-bases. Deploying an effective PKI is said to be more difficult than it sounds. This master thesis implements a distributed firewall for small to medium sized networks. Such a network is not larger than for a system administrator to have responsibility and overview of the whole network. That is the reason PKI is not used. It is discussed as an option to scalability in "further work" in section 10.2.5 on page 97.

Both the clients and the management nodes use a PEM-formatted certificate to set up the SSL connection. Since the function not does any certificate validation, a self-signed certificate is used[14].

[12] PKI is covered by a large number of RFC's. See IETF pkix-charter for a complete list: `http://www.ietf.org/html.charters/pkix-charter.html`

[13] This is why browsers, such as Internet Explorer and Mozilla, ship with a list of CA and their corresponding public keys (or certificates). When browsing on SSL pages signed by one of these CA's, the browser automatically "trust" these and don't annoy the users with questions like "do you trust this host?".

[14] To create self-signed certificate using `openssl`: `openssl req -new -x509 -nodes -out server.pem -keyout server.pem -days 3000`

Chapter 8

Testing

> "To err is human - and to blame it on a computer is even more so."
>
> — *Robert Orben*

In this chapter, various testing is shown. Covering briefly the ping command, then PIDE and some time measures of encryption using blackbox. Then the distributed firewall, using "master" and "client", are shown.

8.1 Ping

Since the ping command behave differently depending on which operating system used, and even different ping exists running on the same operating system, a plattform independent ping is implemented. This ping may function as a standalone program or as a python module.

The ping works as any other normal ping, if run standalone:

```
> ping.py -c 4 10.1.1.2
PING 10.1.1.2 (10.1.1.2): 84 data bytes (20+8+56)
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=1.21188 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.61417 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.62895 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=64 time=0.67115 ms

--- 10.1.1.2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.614/0.782/1.212/0.249 ms
```

And using IPv6:

```
> ping.py -6 -c 4 2001:700:500:49:250:daff:fe4e:758e
PING 2001:700:500:49:250:daff:fe4e:758e : 104 data bytes (40+8+56)
64 bytes from 2001:700:500:49:250:daff:fe4e:758e: icmp_seq=1 time=1.14894 ms
64 bytes from 2001:700:500:49:250:daff:fe4e:758e: icmp_seq=2 time=0.95892 ms
```

```
64 bytes from 2001:700:500:49:250:daff:fe4e:758e: icmp_seq=3 time=0.91696 ms
64 bytes from 2001:700:500:49:250:daff:fe4e:758e: icmp_seq=4 time=0.91219 ms

--- 2001:700:500:49:250:daff:fe4e:758e ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.912/0.984/1.149/0.097 ms
```

But the real usage for the ping script is to be used as a module to test reachability inside the management tools `master`:

```
master> ping viper
viper (123.123.123.123) is alive
master>
```

Here the node "viper" responds to the ICMP echo message sent from the management node. The host viper is reachable and up.

8.2 PIDE

The similar intrusion detection mechanism "AIDE" detected the break-in of several official Debian GNU/Linux servers. The attacker used a sniffed password and installed a root-kit. The root-kit was discovered by the file system integrity checker[1] Lets initialize PIDE on a fresh installed system, and initialize the PIDE database using this stripped down pide.conf:

```
# Where is our database file?
database=file:/var/db/pide.db

# New databases (--init)
database_out=file:/var/db/pide.db.new

# Change this to 'no', or remove it to not gzip output
# (only useful on systems with few CPU cycles to spare)
gzip_dbout=yes

# Lengt of summary when doing --check or --update
#   short  = one line telling whats changed
#   normal = same as above and a list of changed files
#   detail = same as above and a detailed list of change
summary=detail

# Files to monitor
/bin               Binlib
/sbin/             Binlib
/usr/local/bin     Binlib
```

[1]The full report on the Debian compromise: `http://lists.debian.org/debian-announce/debian-announce-2003/msg00003.html`

```
/usr/local/sbin    Binlib

/etc               ConfFiles
/usr/local/etc     ConfFiles

/var/log           Logs
```

Here we monitors the main location of binary- , configuration- and log files on any GNU/Linux and *BSD systems. If the PIDE don't report anything, it simply means no anomalies were found:

```
> pide.py -C -c pide.conf
>
```

That surely was no fun. An attacker will most likely leave some traces, or at least try to hide his traces. If the PIDE database is unavailable to the attacker, as described in section 6.2.3, a properly configured PIDE will most likely detect the compromise. To simulate an attack, I'll install a small rootkit with startup scripts and reset any logs that may contain my traces. If running PIDE with the same configuration file given above, the result is different:

```
> ./pide.py -C -c pide.conf

PIDE found differences between database and filesystem!!

Timestamp: Tue, 28 Sep 2004 09:25:59 +0000

Summary:
Total number of files in database: 1827
Total number of files on system  : 1831
Removed files                    : 5
New files                        : 9
Changed files                    : 9

Missing files:
  /var/log/kern.log
  /var/log/dmesg
  /bin/netstat
  /bin/kill
  /bin/ps

New files:
  /bin/.raxxor
  /bin/.raxxor/kill
  /bin/.raxxor/Makefile
  /bin/.raxxor/netstat
  /bin/.raxxor/ls
  /bin/.raxxor/ps-fixed
  /bin/.raxxor/ps
```

```
  /bin/.raxxor/README
  /etc/rc2.d/S99apachee

Changed files:
  /var/log/messages
  /etc/ld.so.cache
  /etc/passwd
  /bin/ls
  /etc
  /etc/syslog.conf
  /bin
  /etc/shadow
  /etc/rc2.d

Detailed information about changes:
File: /var/log/messages
  Perm  : 33184                                    , 33188
  Inode : 482                                      , 2543
  Gid   : 4                                        , 0
  Size  : 5754                                     , 186

File: /etc/ld.so.cache
  Inode : 129234                                   , 129797
  Size  : 24133                                    , 24610
  Blocks: 48                                       , 56
  Mtime : Sat Apr 17 11:41:05 2004                 , Tue Sep 28 12:39:55 2004
  Ctime : Sat Apr 17 11:41:05 2004                 , Tue Sep 28 12:39:55 2004
  MD5   : b706c1fe1c3c70803418278bcded6961         , 59fd2ef4990f94827c9f9379d5f9978b
  SHA-1 : f80fa6ea0e4635926db19c26ce10f04de45e815d , f08ecff85c41aa8bb0ef1b92a41c254b505db396

File: /etc/passwd
  Size  : 988                                      , 990
  Mtime : Mon Aug 23 09:16:52 2004                 , Tue Sep 28 11:36:24 2004
  Ctime : Mon Aug 23 09:16:52 2004                 , Tue Sep 28 11:36:24 2004
  MD5   : b74bb6c84404d528cf61caa92bc30259         , 1cb31de0a8a810f86509bb647a1b1239
  SHA-1 : 14adec16c1b9e9a466072fa9921b68fbc9a5f380 , f579dabcaf05d2e5ab2874f865ef8f3e5126d96c

File: /bin/ls
  Inode : 175366                                   , 289192
  Mtime : Mon Mar 18 16:10:01 2002                 , Tue Sep 28 11:22:39 2004
  Ctime : Sat Apr  5 18:45:10 2003                 , Tue Sep 28 11:22:39 2004

Directory: /etc
  Mtime : Tue Sep 28 11:02:03 2004                 , Tue Sep 28 12:39:55 2004
  Ctime : Tue Sep 28 11:02:03 2004                 , Tue Sep 28 12:39:55 2004

File: /etc/syslog.conf
  Size  : 1664                                     , 1702
```

```
  Mtime : Thu Jan  3 18:23:03 2002                , Tue Sep 28 11:36:57 2004
  Ctime : Sat Apr  5 18:46:22 2003                , Tue Sep 28 11:36:57 2004
  MD5   : 3882195aa34cd97b8a57120bc8b47f3b        , f4d609edc35496f32f3629def5e0c6ca
  SHA-1 : 9e5218689284ff0e118e394623130b9ac708df72 , f3c184a35f4e16a3965650703181932 89f7e7f29

Directory: /bin
  Lcount: 2                                       , 3
  Mtime : Tue Sep 28 11:22:03 2004                , Tue Sep 28 12:44:11 2004
  Ctime : Tue Sep 28 11:22:03 2004                , Tue Sep 28 12:44:11 2004

File: /etc/shadow
  Size  : 709                                     , 711
  Mtime : Mon Aug 23 09:16:52 2004                , Tue Sep 28 11:36:29 2004
  Ctime : Mon Aug 23 09:16:52 2004                , Tue Sep 28 11:36:29 2004
  MD5   : 20513270c8f4e53801c8873f73a75881        , 08996fa133a130a9526c45a9e936d3fa
  SHA-1 : 48d5b190605d81a6e430d67256136bccd329bde1 , 5624bf05234fbcea7dba70a7dea2729c7c78f69f

Directory: /etc/rc2.d
  Mtime : Sat Apr 17 11:30:25 2004                , Tue Sep 28 11:24:49 2004
  Ctime : Sat Apr 17 11:30:25 2004                , Tue Sep 28 11:24:49 2004
```

Now, this was a lot more fun. Since PIDE is configured to give detailed information, both the short ("Summary"), medium ("Missing", "New" and "Changed") and "Detailed information" is shown. After careful examination of the PIDE report, several new files is located under the suspicious directory `.raxxor/` in the `/bin` directory. This surely isn't a standard directory; it is even a hidden directory. There is also a new file in the startup directory `/etc/rc2.d/S99apachee`. It is located under correct runlevel directory (runlevel 2), but has tried to disguised itself as an Apache webserver startup script. There is also some logfiles and binaries that have been deleted. Both the `shadow` and the `passwd` files, which stores account information, are changed. The syslog configuration file, `syslog.conf`, is also changed.

The conclusion is that the host has been compromised and the attacker has installed a rootkit called "raxxor". The attacker has deleted log files and binaries and included some of his own. The rootkit will (re-) start at boot, since it is included in the startup directory and the attacker has tried to hide his tracks by removing some of the logfiles.

PIDE is started and run inside the master program:

```
master> pide check shadow
Connect..success!
Sending..success!
master>
```

When the command has been sent, the client will execute PIDE and reconnect to the master with the result. If anomalies are found, a medium report is included with the result (as shown above). The result is polled from the IN buffer by executing the `poll` command.

When a compromised host has been detected, the system administrator *may* take down the interface by sending the remote command '`/sbin/ifconfig eth0 down`'. But it can not be guaranteed that it will be enforced, since the attacker may have taken control over the distributed firewall as well. The

system administrator should at least get the host off-line and do a forensic analysis of the break-in. The host should then be re-installed.

8.3 Blackbox

I wrote a small script that created a file with random data with N number of blocks, where N is chosen by the user. No padding was necessary since the size is exactly N blocksizes. By creating a file with 150000 64-bytes blocks, gives a file with the size of 9600000 bytes or 9.1552734375 megabytes. To encrypt a file with Cipher Feedback Mode (CFB), the time was:

```
> time ./blackbox-mod.py -e -p secret77 testfile.div
Saving encrypted file ==> testfile.div.crypt

real    0m16.258s
user    0m13.610s
sys     0m0.320s
```

Encrypting the same file using Cipher Block Chaining is a lot faster:

```
> time ./blackbox-mod.py -e -p secret77 testfile.div
Saving encrypted file ==> testfile.div.crypt

real    0m2.921s
user    0m2.180s
sys     0m0.290s
```

As clearly shown, the Python Cryptographic Toolkit is almost 4 times faster using CBC mode rather than CFB. This is the reason CBC mode is used rather than CFB in blackbox.

Blackbox is used by PIDE to generate encryption key and encrypt the database, and by the master and client to generate encryption key during authentication.

8.4 Master/Client

The client is operating in the background, called "daemon" mode in unix. But the master is interacting with the user through a command line interface (a shell). The shell provides a range of commands, as described in section 6.4 on page 55.

The client is started in "foreground" mode, to easier see what happens. *To enable the client to execute iptables script or command, it must be running with root privileges!* The master can be running as an ordinary user (but must be run as root to use ping - which uses raw sockets). Port 8881, is the default port the distributed firewall uses, but can be changed in either the config file or by using options.

```
> client.py -l 8881
Pusher: doWork: Starting endless loop!
Pusher: doWork: back
```

```
Pusher: doWork: back
Pusher: doWork: back
...
```

Here the "Pusher" is regularly polling the OUT buffer, but since no commands are received, the buffer is empty. Pusher has nothing to do, just reports 'back' meaning it finished polling. The OUT buffer is checked every `CHECK_BUFFER_TIME` which is set to 5 seconds.

When we send some commands to the client from the master, for example an iptables command, it is send immediately:

```
master> binexe 10.1.1.10 /sbin/iptables -A INPUT -p tcp --dport 80 -j DROP
Connect..success!
Sending..success!
master>
```

At the same time, the client produces a lot of information:

```
handle: starting
handle: to many threads?
handle: registering thread... success!
handle: doing parser()
parser: start
TYPE:    2
NAME:    /sbin/iptables
OPTION:    -A INPUT -p tcp --dport 80 -j DROP
CONTENT:
handle: parser() finished
'2'
'/sbin/iptables'
'  -A INPUT -p tcp --dport 80 -j DROP'
''
handle: doing cmd_rep()
handle: cmd_rep() complete
handle: doing execute()
execute: Executing command: /sbin/iptables with options:   -A INPUT -p
tcp --dport 80 -j DROP
execute: output []
execute: pid 1173
execute: running for seconds 1
execute: exit status wait() 0
handle: execute() finished
handle: doing format()
handle: format() finished
handler: result type:  <type 'str'>
<?xml version="1.0">
        <ClientResponse>
           <Name>/sbin/iptables   -A INPUT -p tcp --dport 80 -j DROP</Name>
```

```
            <PID>1173</PID>
            <ExitStatus>0</ExitStatus>
            <Time>1</Time>
            <Output></Output>
          </ClientResponse>

handle: de-registering thread... success!
OUT BUFFER FILLED:
['<?xml version="1.0">\n        <ClientResponse>\n           <Name>/sbin/iptables   -A INPUT -p
THREAD DONE!
Pusher: doWork: data in OUT BUFFER!
Pusher: doConnect: connecting
Pusher: doConnect: auth... success!
Pusher: doConnect: sending
Pusher: doConnect: flushing!
Pusher: doWork: back
Pusher: doWork: back
....
```

Here the client first register the thread an assigns the job to that thread. The command is parsed, and then executed. Output is captured, and a client response XML message is created. After the message is pushed onto the OUT buffer, the thread finishes. Since the OUT buffer is a shared datastructure, it is protected by a mutex. Every thread must hold the lock to access the buffer.

When the Pusher is polling the OUT buffer, it is not empty. It then immediately tries to connect to the management node. When the connection is established, it sends the content and flushes the OUT buffer.

The master's `Listener` class just reads the incoming data from the client and put's it onto the IN buffer. This buffer is polled when the command `poll` is executed (and then flushed):

```
master> poll
REPORT from 10.1.1.10 (10.1.1.10):
       cmd: /sbin/iptables   -A INPUT -p tcp --dport 80 -j DROP
       pid: 1173  with exitstatus: 0 (success!)
       running time: 1
       output: (no output)
master>
```

The same behaviour is shown for every command sent from the management node:

```
master> ping 10.1.1.10
10.1.1.10 (10.1.1.10) is alive
master> fire file block-udp.sh 10.1.1.10
Connect..success!
Sending..success!
master> fire file block-udp.sh 10.1.1.11
Connect..success!
Sending..success!
```

```
master> poll
REPORT from 10.1.1.11 (10.1.1.11):
       cmd: block-udp.sh
       pid: 1052  with exitstatus: 0 (success!)
       running time: 1
       output: (no output)
REPORT from 10.1.1.10 (10.1.1.10):
       cmd: block-udp.sh
       pid: 1368  with exitstatus: 0 (success!)
       running time: 1
       output: (no output)
master>
```

Chapter 9

Other solutions

> "As you know, security is not a product that can be purchased off the shelf, but consists of policies, people, processes, and technology."
>
> — *Kevin Mitnick, Slashdot interview February 5. 2003*

Since I first started thinking and developing the distributed firewall, I've come across several other similar projects. Unfortunate, many (small unsuccessful) open source project is active for a short period of time. This includes firewall projects. I have not been able to test any commercial distributed firewalls, since I've been unable to get hold any copies despite requests[1]. F-Secure's distributed firewall consists of, as far as I can tell, a management node which enforces policies at the end-hosts (workstations running windows). The prototype implemented in this master thesis may run on routers/gateways as well as end-hosts, as long as the hosts support Python and a scriptable firewall.

9.1 STRONGMAN

Steven Bellovin, who was the first to propose a distributed firewall in [5], has been involved in implementing a distributed firewall at the "Scalable Trust of Next Generation Management (STRONGMAN)" project at University of Pennsylvania. The project has developed a prototype of a distributed firewall as specified in the paper "Implementing a Distributed Firewall" [37].

The solution uses KeyNote[2] [9] as policy language, the IPSec key management protocol (IKE) to distributed the policy safely, and a userspace daemon with a kernel-module as policy enforcer at the end-hosts. The prototype is implemented on OpenBSD[3], a free unix operating system, which has a tight integration of security mechanisms as IPSec, OpenSSl and KeyNote.

The implementation includes a kernel extension (available as a loadable module), which modifies two system calls `connect()` and `accept()`. In UNIX, these system calls creates and allows outgoing and incoming TCP connections. Since all users may use these calls, the kernel extension modifies and allows these calls to be "filtered" based on a policy set by the administrator. The user level daemon

[1] The only commercial distributed firewall I've come across is F-Secure's "Distributed Firewall": `http://www.f-secure.com/products/ds-firewall/index.shtml`

[2] KeyNote homepage: `http://www.cis.upenn.edu/~keynote/`

[3] OpenBSD homepage: `http://www.openbsd.org/`

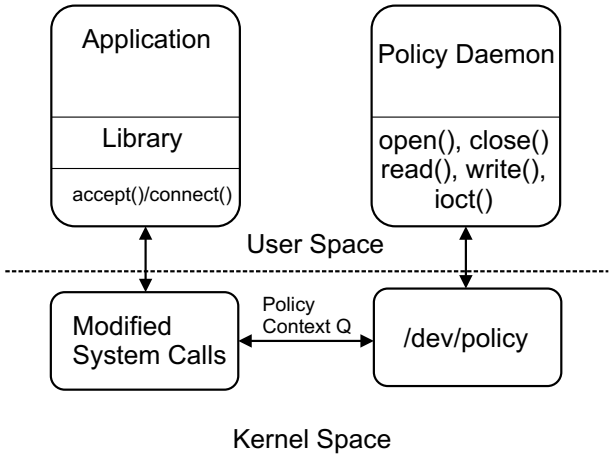

Figure 9.1: The design of the distributed firewall as implemented in [37].

accepts or fetches new polices from the management node. The user level daemon and the kernel extension communicate via a device driver `/dev/policy` as shown in figure 9.1.

As an application often uses TCP to communicate remotely with other application, they use different ports. A port is just a logically entity in the kernel allowing for multiple TCP instances communication over different "ports" at the same time. Certain application have been assigned specific TCP ports; SSH uses port 23, mail uses port 25, web uses port 80 etc.[4]. This way "application filtering" can be done to a certain extent, by blocking or allowing certain ports. This is what the STRONMAN keynote filtering calls "application filtering". Strictly speaking, this is *not* application filtering, but packet filtering. Application level filtering is done at a higher level using some sort of proxy. This sort of filtering does not prevent the application to run on a different port however. This master thesis also uses a packet filter to filter traffic.

STRONGMAN allows for an even more fine grained control. Although not implemented, the paper discusses the possibilities to allow for application filtering based on certain system calls the application makes. A java applet may for example be filtered, but ordinary http traffic may not. I think personally this is a very "ugly" filtering mechanism, as the application could produce a lot of "system call" errors which for the end user will be confusing at best.

This prototype is restricted to open source operating systems, as it requires access to the kernel source to modify system calls. Since the kernel source is not available in proprietary operating systems, this sort of change is not possible. The paper does mention another approach by linking the application to an alternative library with modified system calls. This would make the implementation more operating system independent, as kernel modification is not necessary. But the security depends on the application using the library - which can not be guaranteed. An attacker would certainly *not* use the modified library and thus bypassing the whole firewall.

The STRONGMAN implementation only filters on TCP and other traffic, like IP and UDP, is not filtered. This makes the firewall, in my opinion, very weak. It does mention future extension to include other types of traffic by using built-in packet filtering mechanism, as I uses in my implementation. The STRONGMAN does not include any effective mechanism to prevent DoS attacks

[4]A more comprehensive list can be found by issuing `cat /etc/services` on a UNIX system or consulting the official port number page at IANA: `http://www.iana.org/assignments/port-numbers`

using some form of "rate" control: "This type of packet have I seen 100 times the last 10 seconds, block from source for 3 minutes". The IPTables packet filter allows for this kind of rate control.

The use of IPSec has not yet gained widespread use. It can be a hassle to configure and not all operating systems support it. Using an application level encryption (SSL/TLS) instead of IPSec would avoid these problems. The distributed firewall implemented in this master thesis uses application level encryption.

9.2 Similar projects

I've encountered several similar project, but none which includes strong focus on the *distributed* element of firewall.

9.2.1 Firewall builder

The Firewall Builder[5] `fwbuilder` is a powerful graphical application which lets the user create complex firewall rules and the "compile" it to various scriptable firewalls. As of this writing it compile to ipchains (Linux < 2.4), iptables (Linux 2.4/2.6) and BSD's ipfilter and pf. When the fwbuilder compiles a firewall, it literally translate its own internal XML format into a shell script that when run, initiates the firewall.

The application main focus is to create and craft firewall scripts, but does support a way of remotely distribute a "compiled" firewall. But the way it does this is by executing a SSH session, execute the shell script (trying to use `sudo` if not root). The output of the session is logged to a separate window.

This solution has functional similarities with my solution. Here the script is just executed on the client, output is captured and shown at the "management node". My implementation also just executes and returns any output, although slightly more formatted. No intelligence, except policy enforcing (execution), is present at the client.

9.2.2 Webmin

Webmin[6] is a big and mighty administration tool. It is primarily developed for Linux but run on *BSD as well. It launches its own web-server on a designated port (default 10000) with root privileges. The user then interact with the CGI powered webmin pages. Webmin has a set of core functionality (which includes network configuration, cron jobs, changing password etc) and a large number of plugins, which includes firewall functionality.

The firewall plugin provides a basic configuration of iptables rules, although not as powerful as "Firewall Builder". The web-interface is somewhat simpler than Firewall Builders and there is not "objects" (group of nodes, group of rules) that can be shuffled around (as is the case with Firewall Builder). Most of the iptables conditions are covered, and that is perhaps what most users needs.

Just the idea of having a web-server running with root permission capable of doing remote execution of commands and practically configure anything you want, including shutdown the node, is not a comforting though for a security paranoid mind. Webmin also lacks scalability; the webmin run on

[5] Firewall Builder homepage: `http://www.fwbuilder.org/`
[6] Webmin homepage: `http://www.webmin.com/`

each node. This makes the distributed firewall backwards: All "intelligence" is located at the client through a web-server serving pages over TLS/SSL encrypted https. The "management node" just administer and creates the policy over SSL (HTTPS), which the webmin enforces. Building firewall rules is also a tiresome job, since it must be crafted though the web-interface on each node.

9.2.3 Prelude

Prelude[7] is not a distributed firewall, but a distributed intrusion detection. Prelude has support for several third party sniffers (data collectors) that are called "sensors". Since both network and host based intrusion detection are supported, Prelude is a called a *hybrid* intrusion detection.

All sensors log incidents in a modified "Intrusion Detection Exchange Format (idwg)"[8] XML binary format to a centralized database. This database can then by queried by various front-end tools. Despite being the most active and popular distributed intrusion detection, it has no centralized management to enforce new intrusion detection rules or, as far as I can see, any support for firewall.

[7] Prelude homepage: `http://www.prelude-ids.org/`

[8] IETF "Intrusion Detection Exchange Format" Charter: `http://www.ietf.org/html.charters/idwg-charter.html`

Chapter 10

Conclusion and further work

> "The user's going to pick dancing pigs over security every time."
>
> — *Bruce Schneier*

10.1 Conclusion

In the early days of Internet, it was primarily a place of education and research. Today it is a place where a large amount of firms conduct their daily business. The Internet today has become a critical part of the society's infrastructure. This has also changed the focus on security: From initial focus, from a system administrator's point of view, on technical security mechanisms using encryption, firewalls and intrusion detection. Today, the focus has been lifted to more security awareness with economical, financial and risk management analysis. *"Technical computer security measures such as the use of passwords, biometrics, antivirus software and intrusion detection systems cannot totally reduce an organization's risk to computer security breaches with their associated financial losses"* (CSI/FBI report [29]). But even if the risk never can be eliminated, Schneier says: *"No one can guarantee 100% security. But we can work toward 100% risk acceptance."*[1].

Firewall is one of the oldest and most used security mechanism. But the traditional firewall faces a lot of challenges, and Bellovin was the first to purpose an improved firewall to meet these challenges. His paper "Distributed firewalls" [5] outlined a design for such a distributed firewall. In a distributed firewall, the policy is determined at a centralized management node, then distributed securely, and enforced on the end-nodes (clients). Bellovin's solution met various shortcomings of the traditional firewalls including:

- A firewall may not filter traffic it can not see; internal traffic is something a gateway firewall can't interfere with. The CSI/FBI survey reports that the numbers of successful attacks from the inside are equal to the attacks from the outside (50%-50%). Using a gateway firewall, cover just half the threat. A distributed firewall also covers internal traffic.
- With the introduction of wireless networks, the employees are becoming more mobile. They are no longer restricted to work at the companies physical location to access the Internet. According to [74], 40% of the Norwegian companies provide access to company resources from the outside

[1] Quote taken from Bruce Schneier's essay "Why Cryptography Is Harder Than It Looks" found at `http://www.schneier.com/essay-037.html`

(home/travel). A traditional firewall located at the gateway can not filter and protect mobile host on foreign networks.

- The link speed is increasing rapidly. A modern computer sold today often comes with a gigabit ethernet interface. If enough hosts on the inside generates large amount of traffic, the single gateway firewall will become a network bottleneck since it can't process the packets fast enough. By easing the firewall rules at the gateway and distribute the firewall rules out into the network, this bottleneck is removed.

- Certain protocols, like FTP, are not easily filtered by a firewall located in between. By providing filtering on the end-host, filtering is easier since relevant information (FTP port numbers) is present[2].

- A similar problem is the arrival of end-to-end encryption. The gateway firewall can't filter traffic it don't understand (it *may* drop it, but it can't make the distinction between legetimate and attack data traffic). A mobile node that has been compromised may use VPN to gain access to the company's internal network. Since the data is encrypted, the attack data goes undetected through the gateway firewall. A distributed firewall may block this, as it may filter on encrypted traffic as well.

- An employee may set up an unencrypted access point to be able to surf the net using his new gadget. Even authorized access points are unencrypted, according to [74] 30% of all corporate wireless networks in Norway are open (unencrypted). These "backdoors" are a security risk since they provide crackers with a "free-ticket" into the corporate network. The distributed firewall may be located on every gateway, leaving the attacker at the perimeter of the network and thus limiting the "windows of vulnerabilities".

Despite these shortcomings, the traditional firewall should not to be rendered obsolete. Both [5] and [37] also argue for this. There are several scenarios where the traditional firewall still serve its purpose:

- It is easier to counter active attacks (read more about active attacks under section 2.2.2 on page 11) at the network gateways. When an attack is underway it may be reported (by an intrusion detection) and all traffic from the attacking source is blocked at the gateway. Also, recovering from a DoS attack is more effective at network gateways.

- The distributed firewall may protect hosts that do not support a distributed firewall. That may be embedded gadgets or visiting nodes.

- The traditional firewall may also act as a fail-safe mechanism. If other security mechanism fails, at least *some* basic security is provided by the gateway firewall.

The purpose of this master thesis was, as described in the first chapter, to meet the shortcomings of the traditional firewall by using a distributed firewall. Now it's time to check whether these shortcomings have been met.

As described in section 3.4.2 on page 31, Bellovin listed three components of the distributed firewall.

[2] IPTables even have a module to handle FTP connections, called "ip_conntrack_ftp", that may be used on the end-host.

- **Policy language:** In this master thesis, the policy language are the commands given to a scriptable firewall provided by the operating system. On Linux, this is accomplished using `iptables` commands.
- **System managment:** This is provided by the master and client framework as discussed in section 6.4 on page 55.
- **Safe distribution:** The security policy is distributed securely to the clients using SSL. The mangamement node and client are authenticated using a three way handshake. Both SSL and the authentication mechanism are discussed in section 7.4 on page 71.

The two *main* purposes where to better be able to filter traffic on the internal network, and to be able to protect mobile users. These two main purposes has been met: By using a distributed firewall on all the end-nodes or central gateways on the internal network, inside traffic can be filtered. By running the distributed firewall, all mobile users can also be protected.

Even if this solution limits the threats not covered by the traditional firewall, computer security will be an important part of the Internet in the years to come. Because, as Schneier put's it: *"Network security risks will always be with us. The downside of being in a highly connected network is that we are all connected with the best and worst of society."*[3] And since we can't build absolute secure systems, as Bishop [8] put's it: *"In reality, we cannot yet build systems that are guaranteed to be secured or to remain secure over time"*. Until that time, the only thing that can be done, is to make the job harder for the attacker to succeed. This is exactly what the distributed firewall does.

10.2 Extension and further work

Since I first started developing the distributed firewall, I was forced to cut down on several features originally planned, due to lack of time. Several features, like the EAP-TLS authentication and incorporating network intrusion detection, turned out to be much more challenging to implement than I first anticipated. All of the originally planned features and extensions are listed below, some which is enough work to become a master thesis of its own.

10.2.1 Python Intrusion Detection Environment (PIDE)

PIDE really don't need any extensions. It is fairly robust as it functions right now. These two extensions suggested would give a performance penalty. Since PIDE already is quite CPU intensive, I would be somewhat reluctant to include them without any obvious reason.

Support for databases

PIDE could be extended to store it file information database to a SQL database, and not to a file as it does today. This can be done by creating an abstraction layer in PIDE. The abstraction layer then save to the designated format (either file or database).

[3] Testimony and Statement for the Record of Bruce Schneier: Hearing on Internet Security before the Subcommittee on Science, Technology, and Space of the Committee on Commerce, Science and Transportation: `http://www.schneier.com/testimony-commerce.html`

This would suffer a performance penalty since introducing an extra layer and accessing third party database modules is slower than just dumping the whole database raw to file. It could however be an attractive solution, if the database is located on a remote host, and its fetched over the network using SSL.

XML format

The database today has speed optimized format; each record has a fixed place. This makes parsing very fast, but not very flexible. Adding or removing field must be hard coded. By using a more flexible format, as XML, the database could easily be extended or reduced based on what to monitor. For example, monitor "number of blocks" (`b`) is a Linux specific attribute and is not available on all operating systems. By using XML, each field monitored could be specified in XML and determined and checked at run-time.

By invoking the XML modules and parsing the XML three by using either DOM or SAX, would give a significant performance penalty.

10.2.2 Master and client

Graphical User Interface (GUI)

Several enhancement can be made to the master's framework. Perhaps the most notably, is the incorporation of a graphical user interface (GUI). Using a command line (CLI) shell is not the most intuitively way of interact with the distributed firewall. The master is designed such that the shell can easily be replaced with other interfaces. The command line shell may even co-exists with a GUI providing both a graphical and a CLI way of interaction.

Python has bindings to the most popular graphical libraries. Included with the 'vanilla' Python distribution is a thin object-oriented layer on top of Tcl/Tk[4] called *Tkinter*. Other popular third party graphical libraries like GTK[5], Qt[6] and wxwidgets[7] could be used. The GTK library has improved in performance the last years but is know to be a hassle to get working in a Windows environment. Qt is easier to get working under Windows and is fast, but has a dual licensing making it "semi-free". The wxWidget toolkit has been praised as being fast, highly portable and stable. The creator of Python, Guido van Rossum, has stated that "wxPython is the best and most mature cross-platform GUI toolkit, given a number of constraints. The only reason wxPython isn't the standard Python GUI toolkit is that Tkinter was there first". That's the reason why I'm going to use wxpython to build the graphical user interface.

[4]Tcl (Tool Command Line) is a popular, yet old, programming language which embeds Tk, the graphical user interface toolkit. Read more here: `http://www.tcl.tk/`

[5]GTK (The GIMP toolkit) was originally a graphical library for GUMP, but was later used as graphical library for Gnome: `http://www.gtk.org/` with corresponding Python bindings: `http://www.pygtk.org/`

[6]Qt, the graphical library used in KDE, is developed by the Norwegian firm Trolltech: `http://www.trolltech.com/` with corresponding Python bindings: `http://www.riverbankcomputing.co.uk/pyqt/index.php`

[7]wxWidgets homepage: `http://wxwidgets.org/` with corresponding Python bindings: `http://www.wxpython.org/index.php`

Authentication

The authentication mechanism as it is today can be improved. The SSL connection does not do any key verification (due to Pythons lack of support for this). So the "real" authentication happens inside the encrypted SSL tunnel. This authentication is just a simple three-way handshake.

A more robust well-known authentication mechanism is wanted. The EAP protocol, which is not an authentication protocol in itself, but a protocol optimized for carrying authentication messages, is preferred. EAP has been used for a long time; first it was used as authentication in PPP sessions [2] and later in wireless environments using 802.1X [1] [46]. Inside the EAP protocol, a TLS authentication (EAP-TLS [2]) is to be used. EAP-TLS authenticate both ways using x509 certificates.

But using certificates to handle authentication can in some cases be to ask too much: small networks don't need a full blown PKI. A modular authentication mechanism is preferred, where the implemented handshake can be used on small to medium sized networks, and PKI on large installations. This is discussed under "scalability" below.

Detachment

Like the master program is today, it is only accepting reports from clients when the master is running. When the master is running, it provides some form of interaction to the user. The interaction implemented, is a command line interface (the shell). When the user exits the running shell, it also closes down all the other working threads running in the background (the `Listener` class). So if a client wants to submit a report to the master, it would fail because the master isn't listening. This would normally not be any problem, since the master shell can be started inside a `screen`[8] and detached when for example logging off.

When a GUI is developed, it would be preferably to have separated the GUI from the running master. The GUI would then re-attach itself to the master when starting up. This form of interprocess communication can be performed through a socket or device entry (a `/dev` entry). This way, the master would be running as a daemon in the background performing logging task from clients. These logs are then provided when the system administrator is "logging back on", for example starting up the GUI interface to the master.

Command parser

The commands that can be given from the management node can be any command. It would be preferable to restrict the number of potentially dangerous commands (for example: '`chmod 000 /`' or '`rm -rf /`'). If the management node gets compromised or a dangerous command is entered erroneously, they would get denied by a command parser.

This could be implemented in the `parser` function, that checks each command from a set of "allowed" commands or a set of "denied" commands. If the command is not allowed, or is in the denied list, it is not executed.

[8] The unix program `screen(1)` is a screen manager with terminal emulation. Screen can be detached, and then runs in the background, and later re-attached. Running screen in most handy when running processes on a remote server over an unstable link.

Group management

On of the prioritized task that is going to be implemented in the near future, is the support for group management. Today, commands can only be given to one host at the time. It would be preferable to give commands to a group of hosts at once. For example, to block all web-surfing (TCP source port 80) in one of the classrooms at a school, the administrator could issue the command:

```
master> block all source tcp 80 classrom1A
```

This can be implemented by creating group 'aliases', where the name of the group is the name of the alias file. Within the file, one host is given at each line. So the dispatcher just assigns the same command to each of the hosts specified in the alias file. Some mayor rewriting must be done to support this, and will also support hierarchical rules as described in section 3.4.3 on page 32.

Policy language

As the distributed firewall works today, very few firewall rules are "hard-coded" within the master/client itself. This is due to the potential scripting "mess" all the iptables wrappers needed to be constructed.

To implement a policy language, that is an abstraction level above all operating specific firewall commands, is a huge task. The program "Firewall builder", as discussed in section 9.2.1 on page 89 does this. It would be highly beneficial, since it would make the need for building a very own firewall "library" (firewall shell script) superfluous.

10.2.3 Stat

By implementing a stat module, it could provide the management node some extra information about a client. Data such as the current load, RAM usage, free disk space, uptime, number of users logged could be gathered. This is would be an auxiliary function.

10.2.4 Network Intrusion Detection

The last large feature missing is support for a "Network Intrusion Detection" (NIDS) (read more about IDS in section 4.4.3 on page 37). Including support for NIDS is much work; I see two approaches, either from the client's or the master's view:

1. **The client** software is extended to support interaction with a third party network sniffer. This sniffer can be any network intrusion detection software tool. The open source software Snort is widely used and has had an active development over the years.

   I was thinking of using Snort as a network intrusion detection on the clients. For some well-defined snort reports, the client would generate some well-defined firewall rules. If, for example, a portscan is detected, a firewall rule that blocks all traffic from the source host is executed. The sort of intelligent decision making must be based on strict rules ("this intrusion detection report gives this firewall rule"). These rules can either be hardcoded into the client or dynamically updated by querying the master node for updated rules on a regularly basis.

2. **The master** can also in charge of enforcing firewall rules, without any modification on the client. By using the distributed intrusion detection Prelude, as described in section 9.2.3 on page 90, and let Prelude report directly to the master.

   When Prelude is reporting an incident to the master, the master can distribute firewall rule(s) to the client(s) in question. The decision making is now centralized at the master and not at the client as the solution above. Either approach would require lot of work and lots of rewriting (porting of the framework must be redesigned).

By enabling a network intrusion detection, it would no longer be a distributed firewall but a "distributed firewall using hybrid intrusion detection". The hybrid intrusion detection is here both host based detection, using PIDE, and a network intrusion detection, using either Snort or Prelude.

10.2.5 Scalability

The distributed firewall as implemented today, is supposed to run on small to medium sized network. The firewall tool, *may* be used on large network if each administrator runs it on portion of the network they have responsibility for. To improve the scalability, several enhancements must be implemented.

Authentication

The authentication scheme does not scale very well. Each host's have an entry in the `secrets.db` file listening their shared secret. A more robust solution, like EAP-TLS should be used. EAP-TLS does not only provide a two way authentication, but also keying material that can be used for session encryption. Using EAP-TLS to verify keys also require a PKI, discussed next.

PKI

On large installations, PKI should be used to provide a robust solution to checking certificates during authentication. It could also for example, be used in conjunction with the hierarchical rules as described in section 3.4.3 on page 32, on for example all the schools in a country/county.

All the global rules are set from the central school administration in that country/county. All the schools fetches these global rules regularly (for example every hour), and each school enforces them. This would break the original concept of the management node doing the distribution, as each school (which functions as a "client") asks for the rules. But a scale of this magnitude, it would overload the management node if it should push out the new rules to all the thousands schools. To block all the schools from surfing on porn, a global rule-set can be created that blocks a number of web-sites. This rule-set is effectively enforced on all the schools around the country/country.

Group management

To scale to larger networks, group management, as just discussed, must be implemented. There can also be several system administrators, each having responsibility over portions of the network. The distributed firewall must also support group management of system administrators.

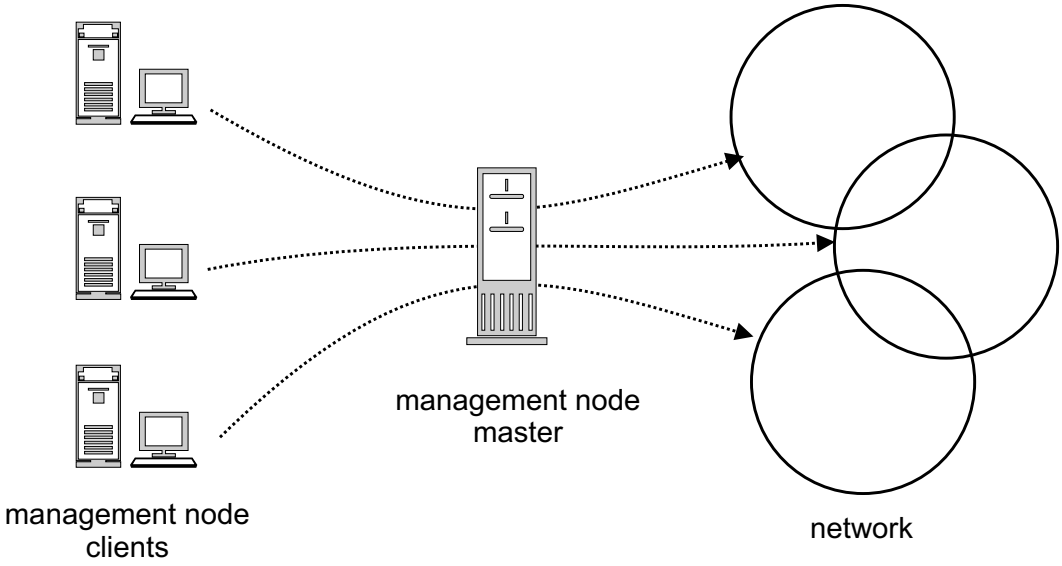

Figure 10.1: Scalability of the management node.

To implement this, I think that the management node must be running as a daemon, and several "management node clients", which is used by the system administrators, connects to the management node each time a new command is given. The management node client takes a command from the system administrator, which connects to a centralized management node, which distributes the commands to all the clients. To implement this, huge design changes must be made. See figure 10.1.

Logging

The last thing that must be changed, are the way logs are handled. Today, all logs are immediately made available to the system administrator and shown when issuing the `poll` command. If there is large amount of clients, and a group of them have received a commands which all have failed to execute (due to a typo from the system administrator), a large amount of logs may overwhelm the management node.

A more intelligent logging facility should be created; for example, the ability to log to file, to be able to group logs together (as the example above, it would group all error logs and report: "All 143 nodes in group X failed to execute command Y").

Appendix A

Source code ping.py

ping.py

```
#!/usr/bin/env python
# -*- coding: iso-8859-1 -*-
"""ping.py

ping.py uses the ICMP protocol's mandatory ECHO_REQUEST
datagram to elicit an ICMP ECHO_RESPONSE from a
host or gateway.

Copyright (C) 2004 - Lars Strand <lars strand at gnist org>

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.

Must be running as root, or write a suid-wrapper. Since newer *nix
variants, the kernel ignores the set[ug]id flags on #! scripts for
security reasons

RFC792, echo/reply message:

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Data ...
```

```
 +-+-+-+-+-
40

TODO:
 - do not create socket inside 'while' (but if not: ipv6 won't work)
 - add support for broadcast/multicast
45 - add support for own payload string

CHANGELOG:
 DONE --> add more support for modules (raise instead of sys.exit)
 DONE --> locale func names
50 DONE --> package def
 DONE --> some code cleanup

"""

55 import sys
import os
import struct
import array
import time
60 import select
import binascii
import math
import getopt
import string
65 import socket

# total size of data (payload)
ICMP_DATA_STR = 56

70 # initial values of header variables
ICMP_TYPE = 8
ICMP_TYPE_IP6 = 128
ICMP_CODE = 0
ICMP_CHECKSUM = 0
75 ICMP_ID = 0
ICMP_SEQ_NR = 0

# Package definitions.
__program__   = 'ping'
80 __version__   = '0.5'
__date__      = '2004/05/10'
__author__    = 'Lars Strand <lars at unik no>'
__licence__   = 'GPL'
__copyright__ = 'Copyright (C) 2004 Lars Strand'
85
def _construct(id, size, ipv6):
    """Constructs a ICMP echo packet of variable size
    """

90  # size must be big enough to contain time sent
    if size < int(struct.calcsize("d")):
        _error("packetsize to small, must be at least %d" % int(struct.calcsize("d")))

    # construct header
95  if ipv6:
        header = struct.pack('BbHHh', ICMP_TYPE_IP6, ICMP_CODE, ICMP_CHECKSUM, \
                             ICMP_ID, ICMP_SEQ_NR+id)
    else:
        header = struct.pack('bbHHh', ICMP_TYPE, ICMP_CODE, ICMP_CHECKSUM, \
100                          ICMP_ID, ICMP_SEQ_NR+id)

    # if size big enough, embed this payload
```

```
    load = "-- IF YOU ARE READING THIS YOU ARE A NERD! --"

    # space for time
    size -= struct.calcsize("d")

    # construct payload based on size, may be omitted :)
    rest = ""
    if size > len(load):
        rest = load
        size -= len(load)

    # pad the rest of payload
    rest += size * "X"

    # pack
    data = struct.pack("d", time.time()) + rest
    packet = header + data          # ping packet without checksum
    checksum = _in_cksum(packet)    # make checksum

    # construct header with correct checksum
    if ipv6:
        header = struct.pack('BbHHh', ICMP_TYPE_IP6, ICMP_CODE, checksum, \
                             ICMP_ID, ICMP_SEQ_NR+id)
    else:
        header = struct.pack('bbHHh', ICMP_TYPE, ICMP_CODE, checksum, ICMP_ID, \
                             ICMP_SEQ_NR+id)

    # ping packet *with* checksum
    packet = header + data

    # a perfectly formatted ICMP echo packet
    return packet

def _in_cksum(packet):
    """THE RFC792 states: 'The 16 bit one's complement of
    the one's complement sum of all 16 bit words in the header.'

    Generates a checksum of a (ICMP) packet. Based on in_chksum found
    in ping.c on FreeBSD.
    """

    # add byte if not dividable by 2
    if len(packet) & 1:
        packet = packet + '\0'

    # split into 16-bit word and insert into a binary array
    words = array.array('h', packet)
    sum = 0

    # perform ones complement arithmetic on 16-bit words
    for word in words:
        sum += (word & 0xffff)

    hi = sum >> 16
    lo = sum & 0xffff
    sum = hi + lo
    sum = sum + (sum >> 16)

    return (~sum) & 0xffff # return ones complement

def pingNode(alive=0, timeout=1.0, ipv6=0, number=sys.maxint, node=None, \
             flood=0, size=ICMP_DATA_STR):
    """Pings a node based on input given to the function.
    """
```

```
    # if no node, exit
    if not node:
        _error("")

    # if not a valid host, exit
    if ipv6:
        if socket.has_ipv6:
            try:
                info, port = socket.getaddrinfo(node, None)
                host = info[4][0]
                # do not print ipv6 twice if ipv6 address given as node
                if host == node:
                    noPrintIPv6adr = 1
            except:
                _error("cannot resolve %s: Unknow host" % node)
        else:
            _error("No support for IPv6 on this plattform")
    else:    # IPv4
        try:
            host = socket.gethostbyname(node)
        except:
            _error("cannot resolve %s: Unknow host" % node)

    # trying to ping a network?
    if not ipv6:
        if int(string.split(host, ".")[-1]) == 0:
            _error("no support for network ping")

    # do some sanity check
    if number == 0:
        _error("invalid count of packets to transmit: '%s'" % str(a))
    if alive:
        number = 1

    # Send the ping(s)
    start = 1; min = 999; max = 0.0; avg = 0.0
    lost = 0; tsum = 0.0; tsumsq = 0.0

    # tell the user what we do
    if not alive:
        if ipv6:
            # do not print the ipv6 twice if ip adress given as node
            # (it can be to long in term window)
            if noPrintIPv6adr == 1:
                # add 40 (header) + 8 (icmp header) + payload
                print "PING %s : %d data bytes (40+8+%d)" % (str(node), \
                                                             40+8+size, size)
            else:
                # add 40 (header) + 8 (icmp header) + payload
                print "PING %s (%s): %d data bytes (40+8+%d)" % (str(node), \
                                                                 str(host), 40+8+size, size)
        else:
            # add 20 (header) + 8 (icmp header) + payload
            print "PING %s (%s): %d data bytes (20+8+%d)" % (str(node), str(host), \
                                                             20+8+size, size)

    # trap ctrl-d and ctrl-c
    try:

        # send the number of ping packets as given
        while start <= number:
            lost += 1 # in case user hit ctrl-c

            # create the IPv6/IPv4 socket
```

```
    if ipv6:
        # can not create a raw socket if not root or setuid to root
        try:
            pingSocket = socket.socket(socket.AF_INET6, socket.SOCK_RAW, \
                                       socket.getprotobyname("ipv6-icmp"))
        except socket.error, e:
            print "socket error: %s" % e
            _error("You must be root (uses raw sockets)" % os.path.basename(sys.argv[0]

    # IPv4
    else:
        # can not create a raw socket if not root or setuid to root
        try:
            pingSocket = socket.socket(socket.AF_INET, socket.SOCK_RAW, \
                                       socket.getprotobyname("icmp"))
        except socket.error, e:
            print "socket error: %s" % e
            _error("You must be root (%s uses raw sockets)" % os.path.basename(sys.argv

    packet = _construct(start, size, ipv6) # make a ping packet

    # send the ping
    try:
        pingSocket.sendto(packet,(node,1))
    except socket.error, e:
        _error("socket error: %s" % e)

    # reset values
    pong = ""; iwtd = []

    # wait until there is data in the socket
    while 1:
        # input, output, exceptional conditions
        iwtd, owtd, ewtd = select.select([pingSocket], [], [], timeout)
        break # no data and timout occurred

    # data on socket - this means we have an answer
    if iwtd:  # ok, data on socket
        endtime = time.time()  # time packet received
        # read data (we only need the header)
        pong, address = pingSocket.recvfrom(size+48)
        lost -= 1 # in case user hit ctrl-c

    # NO data on socket - timeout waiting for answer
    if not pong:
        if alive:
            print "no reply from %s (%s)" % (str(node), str(host))
        else:
            print "ping timeout: %s (icmp_seq=%d) " % (host, start)

        # do not wait if number of ping packet != 1
        if number != 1:
            time.sleep(flood ^ 1)
        start += 1
        continue  # lost a packet - try again

    # examine packet
    # fetch TTL from IP header
    if ipv6:
        # since IPv6 header and any extension header are never passed
        # to a raw socket, we can *not* get hoplimit field..
        # I hoped that a socket option would help, but it's not
        # supported:
        #   pingSocket.setsockopt(IPPROTO_IPV6, IPV6_RECVHOPLIMIT, 1)
```

```
                # so we can't fetch hoplimit..

                # fetch hoplimit
                #rawPongHop = struct.unpack("c", pong[7])[0]

                # fetch pong header
                pongHeader = pong[0:8]
                pongType, pongCode, pongChksum, pongID, pongSeqnr = \
                          struct.unpack("bbHHh", pongHeader)

                # fetch starttime from pong
                starttime = struct.unpack("d", pong[8:16])[0]

            # IPv4
            else:
                # time to live
                rawPongHop = struct.unpack("s", pong[8])[0]

                # convert TTL from 8 bit to 16 bit integer
                pongHop = int(binascii.hexlify(str(rawPongHop)), 16)

                # fetch pong header
                pongHeader = pong[20:28]
                pongType, pongCode, pongChksum, pongID, pongSeqnr = \
                          struct.unpack("bbHHh", pongHeader)

                # fetch starttime from pong
                starttime = struct.unpack("d", pong[28:36])[0]

            triptime  = endtime - starttime # compute RRT
            tsum     += triptime            # triptime for all packets (stddev)
            tsumsq   += triptime * triptime # triptime^2  for all packets (stddev)

            # compute statistic
            if max < triptime: max = triptime
            if min > triptime: min = triptime

            # valid ping packet received?
            if pongSeqnr == start:
                if alive:
                    print str(node) + " (" + str(host) +") is alive"
                else:
                    if ipv6:
                        # size + 8 = payload + header
                        print "%d bytes from %s: icmp_seq=%d time=%.5f ms" % \
                              (size+8, host, pongSeqnr, triptime*1000)
                    else:
                        print "%d bytes from %s: icmp_seq=%d ttl=%s time=%.5f ms" % \
                              (size+8, host, pongSeqnr, pongHop, triptime*1000)

            # do not wait one second if just send one packet
            if number != 1:
                # if flood = 1; do not sleep - just ping
                time.sleep(flood ^ 1) # wait before send new packet

            # the last thing to do is update the counter - else the value
            # (can) get wrong when computing summary at the end (if user
            # hit ctrl-c when pinging)
            start += 1
            # end ping send/recv while

    # if user ctrl-d or ctrl-c
    except (EOFError, KeyboardInterrupt):
        # if user disrupts ping, it is most likly done before
```

```
        # the counter get updates - if do not update it here, the
        # summary get all wrong.
        start += 1
        pass

    # compute and print som stats
    # stddev computation based on ping.c from FreeBSD
    if start != 0 or lost > 0:  # do not print stats if 0 packet sent
        start -= 1              # since while is '<='
        avg = tsum / start      # avg round trip
        vari = tsumsq / start - avg * avg
        # %-packet lost
        if start == lost:
            plost = 100
        else:
            plost = (lost/start)*100

        if not alive:
            print "\n--- %s ping statistics ---" % node
            print "%d packets transmitted, %d packets received, %d%% packet loss" % \
                  (start, start-lost, plost)
            # don't display summary if 100% packet-loss
            if plost != 100:
                print "round-trip min/avg/max/stddev = %.3f/%.3f/%.3f/%.3f ms" % \
                      (min*1000, (tsum/start)*1000, max*1000, math.sqrt(vari)*1000)

    pingSocket.close()

def _error(err):
    """Exit if running standalone, else raise an exception
    """

    if __name__ == '__main__':
        print "%s: %s" % (os.path.basename(sys.argv[0]), str(err))
        print "Try `%s --help' for more information." % os.path.basename(sys.argv[0])
        sys.exit(1)
    else:
        raise Exception, str(err)

def _usage():
    """Print usage if run as a standalone program
    """
    print """usage: %s [OPTIONS] HOST
Send ICMP ECHO_REQUEST packets to network hosts.

Mandatory arguments to long options are mandatory for short options too.
  -c, --count=N    Stop after sending (and receiving) 'N' ECHO_RESPONSE
                   packets.
  -s, --size=S     Specify the number of data bytes to be sent. The default
                   are 56, which translate into 64 ICMP data bytes when
                   combined with the 8 bytes of ICMP header data.
  -f, --flood      Flood ping. Outputs packets as fast as they come back. Use
                   with caution!
  -6, --ipv6       Ping using IPv6.
  -t, --timeout=s  Specify a timeout, in seconds, before a ping packet is
                   considered 'lost'.
  -h, --help       Display this help and exit

Report bugs to lars [at] gnist org""" % os.path.basename(sys.argv[0])


if __name__ == '__main__':
    """Main loop
    """
```

```
    # version control
    version = string.split(string.split(sys.version)[0][:3], ".")
    if map(int, version) < [2, 3]:
        _error("You need Python ver 2.3 or higher to run!")

    try:
        # opts = arguments recognized,
        # args = arguments NOT recognized (leftovers)
        opts, args = getopt.getopt(sys.argv[1:-1], "hat:6c:fs:", \
                                   ["help", "alive", "timeout=", "ipv6", \
                                    "count=", "flood", "packetsize="])
    except getopt.GetoptError:
        # print help information and exit:
        _error("illegal option(s) -- " + str(sys.argv[1:]))

    # test whether any host given
    if len(sys.argv) >= 2:
        node = sys.argv[-1:][0]   # host to be pinged
        if node[0] == '-' or node == '-h' or node == '--help' :
            _usage()
    else:
        _error("No arguments given")

    if args:
        _error("illegal option -- %s" % str(args))

    # default variables
    alive = 0; timeout = 1.0; ipv6 = 0; count = sys.maxint;
    flood = 0; size = ICMP_DATA_STR

    # run through arguments and set variables
    for o, a in opts:
        if o == "-h" or o == "--help":    # display help and exit
            _usage()
            sys.exit(0)
        if o == "-t" or o == "--timeout": # timeout before "lost"
            try:
                timeout = float(a)
            except:
                _error("invalid timout: '%s'" % str(a))
        if o == "-6" or o == "--ipv6":    # ping ipv6
            ipv6 = 1
        if o == "-c" or o == "--count":   # how many pings?
            try:
                count = int(a)
            except:
                _error("invalid count of packets to transmit: '%s'" % str(a))
        if o == "-f" or o == "--flood":   # no delay between ping send
            flood = 1
        if o == "-s" or o == "--packetsize":  # set the ping payload size
            try:
                size = int(a)
            except:
                _error("invalid packet size: '%s'" % str(a))
        # just send one packet and say "it's alive"
        if o == "-a" or o == "--alive":
            alive = 1

    # here we send
    pingNode(alive=alive, timeout=timeout, ipv6=ipv6, number=count, \
             node=node, flood=flood, size=size)
    # if we made it this far, do a clean exit
    sys.exit(0)
```

```
### end
```

Appendix B

Source code pide.py

pide.py

```
#!/usr/bin/env python
# -*- coding: iso-8859-1 -*-
"""pide.py

PIDE (Python Intrusion Detection Environment) is an intrusion detection
system for checking the integrity of directories and files. Basically
a Python implementation of Tripwire/AIDE. Can read AIDE/Tripwire config
files, but does not make AIDE/Tripwire compatible databases..

Copyright (C) 2004 - Lars Strand <lars strand at gnist org>

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.

http://www.gnu.org/copyleft/gpl.html


TODO:
* xml data format --> performance penalty

Changelog:
 DONE --> encrypt database
 DONE --> locale func names
 DONE --> more gracefule exception handling when used as module
 DONE --> warn dead symlinks! --> refixed Okt2004
 DONE --> stats on directories as well
 DONE --> +S (ok if size is the same or growing) --> gives a small
          preformance penalty, since more cheks has to be added
 DONE --> access by name! not in tuple
          http://mail.python.org/pipermail/patches/2001-December/006887.html
 DONE --> add group of files - added globbing
 DONE --> compressed database
 DONE --> +b (add number of blocks)
 DONE --> if rules is only letters (the same as +) - REVERTED!!
 DONE --> rule-options ==> binLib-p-sha1
 DONE --> degree of summary: short, normal, detail
 DONE --> +sha1 or -sha1 also is interpreted as +a etc..
```

```
          attributes MUST have + or - in front..
 DONE --> add more template rules
 DONE --> verbose setting
 DONE --> check for save new db and old db BEFORE doing any stats!
 SHOULD BE OK NOW - done some sanity checks on =, $ and *!
 DONE --> version control (must have at least python ver. 2.3)
"""

# global modules
import getopt
import sys
import os
import os.path
import re
import glob
import stat
import md5
import sha
import tempfile
import time
import gzip
import string
# try to import the encryption module
try:
    sys.path.insert(0, '.')
    import blackbox
    have_blackbox = 1
except:
    have_blackbox = 0

# Package definitions.
__program__   = 'PIDE'
__version__   = '0.7'
__date__      = '2004/22/03'
__author__    = 'Lars Strand <lars at unik no>'
__licence__   = 'GPL'
__copyright__ = 'Copyright (C) 2004 Lars Strand'

########################## _makeSysDB
def _makeSysDB(buildpaths, confvar):
    """We get a dictionary of 'file => att' and a list of
    attributes.
    for each file:
    - make stats according to 'att'
    - save stats to db

    returns db
      """

    sysDB = {}

    # for each file
    for file in buildpaths:
        attributes = buildpaths[file]
        if verbose: print "stat:", file,

        # The stats datastructure:
        # filename => [stat1, stat2, stat3, .....], filename2 => [stat1. stat2...]
        # file = [perm, inode, lcount, uid, gid, size, blocks, atime, mtime, ctime, md5, sha1]
        sysDB[file] = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

        # stat the file, print warning if dead symlink
        if os.path.islink(file) and confvar['warn_dead_symlinks'] == 'yes':
            try:
```

```
            filemode = os.stat(file)
        except: # the stat will fail if dead symlink
            print "WARNING: Dead symlink? %s" % file

    elif os.path.isfile(file) or os.path.isdir(file):
        try:
            filemode = os.stat(file)
        except:
            print "WARNING: Unable to stat '%s'" % file

    # extract and put the stat in the right place
    # NB! if the rule contain both + and -, the + has precedence
    # +S (check for growing size) is not here, must do a check
    # of this size and the old one.

    # The rest of the stats
    if '+p' in attributes:           # permission and file mode bits
        if verbose: print "+p",
        sysDB[file][0] = filemode.st_mode
    if '+i' in attributes:           # inode number
        if verbose: print "+i",
        sysDB[file][1] = filemode.st_ino
    if '+n' in attributes:           # number of (hard) links
        if verbose: print "+n",
        sysDB[file][2] = filemode.st_nlink
    if '+u' in attributes:           # user id of owner
        if verbose: print "+u",
        sysDB[file][3] = filemode.st_uid
    if '+g' in attributes:           # group id of owner
        if verbose: print "+g",
        sysDB[file][4] = filemode.st_gid
    if '+s' or '+S' in attributes:   # size of file
        if verbose and '+s' in attributes: print "+s",
        if verbose and '+S' in attributes: print "+S",
        sysDB[file][5] = filemode.st_size
    if '+b' in attributes:           # number of blocks
        if verbose: print "+b",
        sysDB[file][6] = filemode.st_blocks
    if '+a' in attributes:           # access timestamp
        if verbose: print "+a",
        sysDB[file][7] = filemode.st_atime
    if '+m' in attributes:           # modification timestamp
        if verbose: print "+m",
        sysDB[file][8] = filemode.st_mtime
    if '+c' in attributes:           # inode creation timestamp
        if verbose: print "+c",
        sysDB[file][9] = filemode.st_ctime

    if os.path.isfile(file):
        if '+md5' in attributes:         # MD5 signature
            if verbose: print "+md5",
            # open the file
            try:
                f = open(file, 'rb')
            except:
                print "Unable to open %s for MD5 hash" % file
                continue

            hash = md5.new()

            # read the file in chuncks
            while 1:
                r = f.read(8096)
                if not r:
```

```
                        break
                    hash.update(r) # feed the hash object

                f.close()

                sysDB[file][10] = hash.hexdigest()

        if os.path.isfile(file):
            if '+sha1' in attributes:       # SHA1 signature
                if verbose: print "+sha1",
                # open the file
                try:
                    f = open(file, 'rb')
                except:
                    print "Unable to open %s for SHA-1 hash" % file
                    continue

                hash = sha.new()

                # read the file in chuncks
                while 1:
                    r = f.read(8096)
                    if not r:
                        break
                    hash.update(r) # feed the hash object

                f.close()

                sysDB[file][11] = hash.hexdigest()

        # some space
        if verbose: print ""

    return sysDB

########################## _buildPaths
def _buildPaths(rcfile):
    """Read the config file:
    * read all config variables (if any)
    * read all rules (if any give) - a set of default rules listed
    * read files (os.walk)

    Returns a 'files' (dict) of 'files => attributs' and
    confvar (dict) of 'variablename => value'
    """

    # read the config file
    try:
        ifile = open(rcfile, 'r')
    except:
        _error("Unable to open config file: %s" % rcfile)

    # read the config file

    # grep '#' lines
    comment   = re.compile(r"\s*\#.*")
    # blank lines
    blanks    = re.compile(r"^$")

    # database
    dbfile     = r"\s*database=file:(\S+)"
    dbfilenew  = r"\s*database_out=file:(\S+)"
    dbzip      = r"\s*gzip_dbout=(\S+)"
    dbcrypt    = r"\s*encrypt_db=(\S+)"
```

```
    dbcryptkey = r"\s*password=(\S+)"
    dbwarnsym  = r"\s*warn_dead_symlinks=(\S+)"

    # crypt add - public key or password?

    # length of summary
    summary   = r"\s*summary=(\S+)"

    # dictionary to hold optional variables
    # fill inn some default variables
    confvar   = {
        'database'           : '/var/db/pide/pide.db',
        'database_out'       : '/var/db/pide/pide.db.new',
        'gzip_dbout'         : 'yes',
        'encrypt_db'         : 'no',
        'warn_dead_symlinks' : 'no',
        'summary'            : 'normal'
        }


    # all defined enviroment variables get exported (for us with cron)
    # @@define MAILTO to@address.org
    envvar    = re.compile(r"\s*@@define\s+(\S+)\s+(\S+)")

    # undef enviroment variables
    # @@undef VAR
    uenvvar   = re.compile(r"\s*@@undef\s+(\S+)")

    # custom rules:
    # Binlib = +p+i+n+u+g+s+b+m+c+md5+sha1
    rule      = re.compile(r"\s*(\S+)\s*=\s*(.+)")

    #    p :  permission and file mode bits      a: access timestamp
    #    i :  inode number                       m: modification timestamp
    #    n :  number of links (ref count)        c: inode creation timestamp
    #    u :  user id of owner                 md5: MD5 signature
    #    g :  group id of owner              tiger: tiger signature  (not implemented)
    #    s :  size of file                  rmd160: RMD160 signature (not implemented)
    #    b :  number of blocks (Linux)        sha1: SHA1 signature
    #    S :  check for growing size
    #

    rules     = {
        'R'         : '+p+i+n+u+g+s+b+m-a+md5+tiger+rmd160+sha1-a',
        'L'         : '+p+i+n+u+g-s-b-a-m-md5-tiger-rmd160-sha1',
        'N'         : '+p+i+n+u+s+g+s+b+a+m+c+md5+tiger+rmd160+sha1',
        'E'         : '-p-i-n-u-s-g-s-b-a-m-c-md5-tiger-rmd160-sha1',
        '>'         : '+p+u+g+i+n+S',
        'Logs'      : '+p+u+g+i+n+S',
        'Binlib'    : '+p+i+n+u+g+s+b+m+c+md5+sha1',
        'ConfFiles' : '+p+i+n+u+g+s+b+m+c+md5+sha1',
        'Devices'   : '+p+i+n+u+g+s+b+c',
        'Databases' : '+p+n+u+g',
        'StaticDir' : '+p+i+n+u+g',
        'ManPages'  : '+p+i+n+u+g+s+b+m+c+md5+sha1'
        }


    # '!' signifies the entry is to be pruned (inclusive) from
    # the list of files to be scanned.
    # !/root/.bash_history
    notfile   = re.compile(r"\s*!(\S+)")
    notfiles  = [] # files/directories which are pruned
```

```
    # The files/directories
    # =/boot$ Binlib
    # /bin binlib
    file      = re.compile(r"\s*(\S+)\s+(\S+)")
    files     = {}

    linenumber = 0

    # for each line in config file
    if verbose: print "Reading config file, line by line"
    for line in ifile:
        linenumber += 1

        # match here, so we can extract var later
        commentmatch   = re.search(comment, line)
        blankmatch     = re.search(blanks, line)
        dbfilematch    = re.search(dbfile, line)
        dbfilenewmatch = re.search(dbfilenew, line)
        dbzipmatch     = re.search(dbzip, line)
        dbwarnsymmatch = re.search(dbwarnsym, line)
        dbcryptmatch   = re.search(dbcrypt, line)
        dbcryptkeymatch= re.search(dbcryptkey, line)
        summarymatch   = re.search(summary, line)
        envvarmatch    = re.search(envvar, line)
        uenvvarmatch   = re.search(uenvvar, line)
        rulesmatch     = re.search(rule, line)
        notfilematch   = re.search(notfile, line)
        filesmatch     = re.search(file, line)

        # skip all comments and blank lines
        if commentmatch:
            continue
        if blankmatch:
            continue
        # where should the db be read from?
        elif dbfilematch:
            database = dbfilematch.group(1)
            confvar['database'] = database
        # where should the new db be saved?
        elif dbfilenewmatch:
            databaseout = dbfilenewmatch.group(1)
            confvar['database_out'] = databaseout
        # should db be zipped?
        elif dbzipmatch:
            dbgzip = dbzipmatch.group(1)
            if dbgzip == 'no':
                confvar['gzip_dbout'] = 'no'
        # warn on dead symlinks?
        elif dbwarnsymmatch:
            if dbwarnsymmatch.group(1) == 'yes':
                confvar['warn_dead_symlinks'] = 'yes'
        # encrypt database?
        elif dbcryptmatch:
            if dbcryptmatch.group(1) == 'yes':
                confvar['encrypt_db'] = 'yes'
        # encrypt password
        elif dbcryptkeymatch:
            try:
                confvar['password'] = dbcryptkeymatch.group(1)
            except:
                _error('ERROR: Unable to read password from config file!')

        # length of summary: valid are 'short', 'normal' and 'detail'
        elif summarymatch:
```

```
            summaryout = summarymatch.group(1)
            if summaryout == 'short' or summaryout == 'normal' or summaryout == 'detail':
                confvar['summary'] = summaryout
            else:
                print "WARNING: unrecognized summary value '%s' on line: %s" % \
                      (summaryout, linenumber)
        # any evniroment variables? (for cron)
        elif envvarmatch:
            # typcial format: MAILTO, someone@localhost
            os.putenv(envvarmatch.group(1), envvarmatch.group(2))
        # undef any enviromentvariables
        elif uenvvarmatch:
            os.unsetenv(uenvvarmatch.group(1))
        # own defined rules
        elif rulesmatch:
            rules[rulesmatch.group(1)] = rulesmatch.group(2)
        # files to exclude - remove these files from files{} when loop complete
        elif notfilematch:
            # it the line ends with '$', remove it.
            # used to express the end of a path
            #  /tem$  == mening /tem and NOT /temm or /temp or /te ...
            # but the glob function in python does that automatic, and the
            # glob function don't like '$', so we remove it
            if notfilematch.group(1)[-1:] == "$":
                for i in glob.glob(notfilematch.group(1)[:-1]):
                    notfiles.append(i)
            else:
                for i in glob.glob(notfilematch.group(1)):
                    notfiles.append(i)

        # get files/directories
        elif filesmatch:

            # se comment for notfilematch - the same thing goes here
            if filesmatch.group(1)[-1:] == "$":
                file2 = filesmatch.group(1)[:-1] # do not include '$' at end
            else:
                file2 = filesmatch.group(1)      # has no '$' at end

            # for hver fil, sjekk om group2 = en rulesmatch
            # så legg til
            todo  = filesmatch.group(2)

            # if using a template (R/L/N/E/>), translate to the
            # corresponding "rules"
            if rules.has_key(todo):
                todo = rules[todo]
            # NB! Rules may also be used as
            #   BinLib-md5-sha1
            # Where 'BinLib' is a rule and we don't want md5 and sha1
            # The above test will fail, and we need to split the 'todo'
            # with '+' and/or '-' as delimiter
            else:
                # example: 'BinLib-p-a-c+md5 -s'
                # first we must get 'BinLib' - fetch that using regexp
                r = re.search(r"^(\w+)(.*$)" ,todo)  # matches (text+numbers)(the rest)
                todoo = ""                           # to hold the translated rule
                if r:                                # if match
                    if rules.has_key(r.group(1)):    # is it a know rule?
                        todoo = rules[r.group(1)]    # translate rule for later user
                    else:                            # unknow rule!
                        print "WARNING: Unknown rule on line %d! Ignoring" % linenumber
                        continue
```

```
                    # ok we now have translated the rule, add the rest if any
                    if r.group(2):                # more than just a rule?
                        rulessplit = string.split(str(r.group(2))) # first we split on blanks
                        for g in rulessplit:      # add the rest
                            todoo += g
                        # NB! The rule may now contain both -p and +p
                        # but when doing stats, the +p has presedence
                    # add rule:
                    todo = todoo

            # '=' signifies the entry is to be added, but if it's
            # a directory, then all its contents are pruned (useful for /tmp)
            # No problem with glob.
            if verbose: print "Building pathlist: %s" % file2
            if file2[0] == '=':
                for f in glob.glob(file2[1:]):
                    files[f] = todo  # do not include '='

            # ok, add the file/directory
            else:
                for f in glob.glob(file2):
                    if os.path.isfile(f):   # if file - just add
                        files[os.path.join(f)] = todo
                    elif os.path.isdir(f) or os.path.islink(f):   # if dir - do a walk
                        for root, dirs, file3 in os.walk(f):
                            for i in file3: # for all files in dir
                                files[os.path.join(root, i)] = todo
                            if root: # add directory as well
                                files[root] = todo
        else:
            print "WARNING: unrecognized line: %s" % line

    # exclude the files from notfiles
    if verbose: print "Excluding files, if any.."
    for i in notfiles:
        if files.has_key(i):
            del files[i]

    # Ok, we have dictionary of files in 'files' and a list of
    # variables in 'confvar'
    return files, confvar

########################## _saveDB
def _saveDB(sysDB, confvar, password, crypt):
    """Save the database to file, given in 'database_out'. Compress
    the database if compress is true
    """

    # where should the database be saved?
    # there is always a (default) value here
    dboutfilename = confvar['database_out']

    # should we compress the database? (default = yes)
    if confvar.has_key('gzip_dbout'):
        if confvar['gzip_dbout'] == 'yes':       # We should gzip the database
            try:                                 # open the dbout file
                dbout = gzip.GzipFile(dboutfilename, 'w')
            except:
                _error("Unable to write to databasefile file: %s" % dboutfilename)
        else:
            # open the dbout file
            try:
                dbout = open(dboutfilename, 'w')
            except:
```

```
                _error("Unable to write to databasefile file: %s" % dboutfilename)

    # write to file
    dbout.write("# This file was generated by %s, version %s\n" % \
                (__program__, __version__))
    dbout.write("# Time of generation: %s\n" % \
                time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmtime()))
    dbout.write("# @@db_spec name perm inode size blocks lcount uid \
    gid size atime mtime ctime md5 sha1\n")

    # write all the stats to file:
    for file in sysDB:                    # for each file in dictionary
        filestats = file                  # contruct a string
        for stat in sysDB[file]:          # extract all stats..
            filestats += ' ' + str(stat)  # ..and append to string
        dbout.write(filestats+'\n')       # write to file

    # close the file
    dbout.close()

    # do encrypt
    if crypt or confvar['encrypt_db'] == 'yes':
        # initiate Reactor class
        crpt = blackbox.Reactor(password, 'AES')
        # do encrypt
        cryptfile = crpt.encrypt(dboutfilename, blackbox.FILE)
        # save to file
        try:
            ofile = open(dboutfilename, 'w')
            ofile.write(cryptfile)
            ofile.close()
        except:
            _error("ERROR! Unable to write encrypted file! %s" % dboutfilename)

    print "Database saved in %s" % dboutfilename

############################ readSysDB
def _readSysDB(confvar, password):
    """Read a pide database from file and put into a dictionary:
    {file : [stat1, stat2, stat3....], file2......}
    return the dictionary
    """

    dbinfilename = confvar['database']
    dbdecrypted = dbinfilename + '.decrypted'

    # encrypted database?
    if crypt or confvar['encrypt_db'] == 'yes':
        # initiate Reactor class
        crpt = blackbox.Reactor(password, 'AES')
        try:
            size = os.path.getsize(dbinfilename)
            ifile = open(dbinfilename, 'rb')
            cryptfile = ifile.read(size)
            ifile.close()
        except:
            _error("ERROR! Unable to read databasefile!")

        # do deencrypt
        cryptfile = crpt.decrypt(cryptfile)

        # save to file
        try:
            ofile = open(dbdecrypted, 'w')
```

```
            ofile.write(cryptfile)
            ofile.close()
        except:
            _error("ERROR! Unable to write encrypted file! %s" % dbdecrypted)

        dbinfilename = dbdecrypted

    # if gzip_dbout == yes, we assume the db file is compressed
    if confvar['gzip_dbout'] == 'yes':  # open using gzip
        try:
            dbin = gzip.GzipFile(dbinfilename, 'r')
        except:
            _error("Unable to read database file: %s" % dbinfilename)
    else:
        try:
            dbin = open(dbinfilename, 'r')
        except:
            _error("Unable to read database file: %s" % dbinfilename)

    # This is where we put our database
    fileDB = {}
    linenumber = 0

    # grep '#' lines
    comment   = re.compile(r"\s*\#.*")
    # blank lines
    blanks    = re.compile(r"^$")

    # read file line by line
    if verbose: print "Reading database from disk, line by line"

    while 1:
        linenumber += 1  # in case of an error, print the line number
        line = dbin.readline()
        if not line: break

        # process line
        if re.search(comment, line):  # skip comment lines
            continue
        elif re.search(blanks, line): # skip blank lines
            continue

        # split the line
        linesplit = string.split(line)

        # read arguments into a dictionary
        if len(linesplit) != 13:           # each line has exactly 13 arguments!
            _error("Error occured reading database '%s' on line %d\nThe offending line was:\n%s
                   (dbinfilename, linenumber, line))
        else:
            fileDB[linesplit[0]] = linesplit[1:]

    # delete unencrypted file - if using encryption
    if crypt or confvar['encrypt_db'] == 'yes':
        try:
            os.remove(dbinfilename)
        except:
            pass

    return fileDB

############################ _compareDB
def _compareDB(called, sysDB, fileDB, confvar, syspaths):
    """Should compare two databases: One (old) from disk, and the newly
```

```
    generated. The databases are two dictionaries.
    * sysDB = new generated database
    * fileDB = (old) database from disk
    """

    missing = []    # files that are in old db, but not in new
    changed = []    # files that have changed since last check
    added   = []    # new files not in old db

    for key in fileDB:
        #try:                        # try in case there is no exceptions --> catch keyerrors
        if not sysDB.has_key(key):      # check if we have the file in the new one
            missing.append(key) # add missing file to list
        #except:                     # keyerror, just..
        #    continue                # ..continue with next file

        if sysDB.has_key(key):              # file exists, check stats
            i = 0                   # to iterate stats
            if verbose: print "Compare: %s" % key
            # for each stat in old db
            for stat in fileDB[key]:
                # compare each stat in old DB with the corresponding stat in new DB (sysDB)
                if str(stat) != str(sysDB[key][i]): # do we have a difference?
                    # must do a exception on '+S'
                    if i == 5 and '+S' in syspaths[key]:
                        if int(fileDB[key][i]) > int(sysDB[key][i]):
                            if verbose: print "   Difference found!"
                            # add the file, it it's not added already
                            if not key in changed:
                                if verbose: print "   File added!"
                                changed.append(key) # add changed file to list

                    else:
                        if verbose: print "   Difference found!"
                        # add the file, it it's not added already
                        if not key in changed:
                            if verbose: print "   File added!"
                            changed.append(key) # add changed file to list
                i += 1

    # for each file NOT old db is considered 'new' files
    for key in sysDB:
        if not fileDB.has_key(key):
            added.append(key)

    # if init and no changes - just return
    if called == "check" and not missing and not changed and not added:
        return
    else:
        # print summary - short/normal/detail get this summary
        print "\n%s found differences between database and filesystem!!" % __program__
        print "\nTimestamp: %s" % time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmtime())
        print "\nSummary:"
        print "Total number of files in database: %s" % len(fileDB)
        print "Total number of files on system  : %s" % len(sysDB)
        print "Removed files                    : %s" % len(missing)
        print "New files                        : %s" % len(added)
        print "Changed files                    : %s" % len(changed)

        # normal/detail summary level
        if confvar['summary'] == 'normal' or confvar['summary'] == 'detail':
            # viewing the files that are missing/new/changed
            if called == "update":            # When called as 'update'..
                if len(missing) != 0:
```

```
                print "\nRemoved files:"   # ...some syntax diff
                for file in missing:
                    print "  " + file
            if len(added) != 0:
                print "\nAdded files:"
                for file in added:
                    print "  " + file

        if called == "check":              # When called as 'check'..
            if len(missing) != 0:
                print "\nMissing files:"   # .. some syntax diff
                for file in missing:
                    print "  " + file
            if len(added) != 0:
                print "\nNew files:"
                for file in added:
                    print "  " + file

        # changed files
        if len(changed) != 0:
            print "\nChanged files:"
            for file in changed:
                print "  " + file

    # detail summary level
    if confvar['summary'] == 'detail':
        print "\nDetailed information about changes:"

        for file in changed:               # for each changed file
            if os.path.isdir(file):
                print "Directory:", file
            else:
                print "File:", file

            i = 0
            for stat in fileDB[file]:     # iterate stats
                if str(stat) != str(sysDB[file][i]):
                    # perm, inode, lcount, uid, gid, size, atime, mtime, ctime, md5, sha1
                    if i == 0:
                        print "  Perm  : %-40s , %-40s" % (str(stat), str(sysDB[file][i]))
                    elif i == 1:
                        print "  Inode : %-40s , %-40s" % (str(stat), str(sysDB[file][i]))
                    elif i == 2:
                        print "  Lcount: %-40s , %-40s" % (str(stat), str(sysDB[file][i]))
                    elif i == 3:
                        print "  Uid   : %-40s , %-40s" % (str(stat), str(sysDB[file][i]))
                    elif i == 4:
                        print "  Gid   : %-40s , %-40s" % (str(stat), str(sysDB[file][i]))

                    # must do some exceptions on +S
                    if i == 5 and '+S' in syspaths[file] and \
                           int(fileDB[file][5]) > int(sysDB[file][5]):
                        print "  Size  : %-40s , %-40s" % (str(stat), str(sysDB[file][i]))
                    elif i == 5:
                        print "  Size  : %-40s , %-40s" % (str(stat), str(sysDB[file][i]))

                    # back to "normal" check
                    if i == 6:
                        print "  Blocks: %-40s , %-40s" % (str(stat), str(sysDB[file][i]))
                    elif i == 7:
                        print "  Atime : %-40s , %-40s" % (str(time.ctime(int(stat))), \
                                                          str(time.ctime(int(sysDB[file][i
                    elif i == 8:
                        print "  Mtime : %-40s , %-40s" % (str(time.ctime(int(stat))), \
```

```
                                                    str(time.ctime(int(sysDB[file][i
                        elif i == 9:
                            print "  Ctime : %-40s , %-40s" % (str(time.ctime(int(stat))), \
                                                    str(time.ctime(int(sysDB[file][i
                        elif i == 10:
                            print "  MD5   : %-40s , %-40s" % (str(stat), str(sysDB[file][i]))
                        elif i == 11:
                            print "  SHA-1 : %-40s , %-40s" % (str(stat), str(sysDB[file][i]))

                    i += 1

                print "" # some space between each file


############################ integrityCheck
def integrityCheck(init=0, check=0, update=0, rcfile="", verbose=0, password='', crypt=0):
    """Starting point - call this if using as a module.
    * call 'buildPaths' which returns a list of files (with attributes)
      and a list of config variables
    * then call 'makeSysDB' to build a database of all files in 'files'
    * choice:
       - init:   save db and quit
       - check:  compare db on disk and the db generated by 'makeSysDB'
                 print changes
       - update: compare db on disk and the db generated by 'makeSysDB'
                 print updates - save new db on disk
    """

    # read the config file, returns a list of files in syspaths
    # and a list of config variables in 'confvar'
    if rcfile:
        if verbose: print "Parsing config file and building paths"
        syspaths, confvar = _buildPaths(rcfile)
    else:
        _error("No config file defined")

    # if crypt and password given at command line - overrid pide.conf
    if password:
        if password < 8:
            _error("ERROR: Password must be longer than 8 char!")
        else:
            confvar['encrypt_db'] = 'yes'
            confvar['password'] = password

    # Do some sanity checks before going into some heavy processing
    if init: # can we open db out for writing? If not there is no need to process files..
        try:
            dbout =  open(str(confvar['database_out']), 'w')
            dbout.close()
        except:
            _error("Unable to write to databasefile file: %s" % confvar['database_out'])

    if check or update: # can we open db for reading? If not we can't compare..
        try:
            dbout = open(str(confvar['database']), 'r')
            dbout.close()
        except:
            _error("Unable to read to databasefile file: %s" % confvar['database'])

    # build database sysDB = database of current system
    # this can be quite large (several MB)
    if verbose: print "Building database of current system"
    sysDB = _makeSysDB(syspaths, confvar)
```

```
    # init = save database as defined in 'database_out', gzip if specified
    if init:
        if verbose: print "Doing 'init' --> saving database"
        # save database to file
        _saveDB(sysDB, confvar, confvar['password'], crypt)

    # update = print the different between the new database (sysDB) and
    # the old one. Then save the new database in 'database'
    elif update:
        # read database from file
        if verbose: print "Reading database from file"
        fileDB = _readSysDB(confvar, confvar['password'])
        # compare the two databases
        if verbose: print "Compare database of system with (old) database on disk"
        _compareDB("update", sysDB, fileDB, confvar, syspaths)
        # save the new database sysDB
        if verbose: print "Save database to file"
        _saveDB(sysDB, confvar)

    # check = compare the new database 'sysDB' with the old one,
    # print the differenses
    elif check:
        # read database from file
        if verbose: print "Reading database from file"
        fileDB = _readSysDB(confvar, confvar['password'])
        # compare the two databases
        if verbose: print "Compare database of system with (old) database on disk"
        _compareDB("check", sysDB, fileDB, confvar, syspaths)

    return 0
    # end

########################## printconf
def printconf():
    """Print an example config file and exit
    """

    print """#
# pide.conf based on Tripwire's tw.config
#
# NOTE! the tiger and rmd160 checksum is NOT implemented! So it
# it will be ignored (but md5 and sha1 should do the job).
#
# This file contains a list of files and directories that this
# script will scan. Information collected from these files will be
# stored in a database file.
#
# Format: [!|=] entry [attributes-flags]
#
# where:  '!' signifies the entry is to be pruned (inclusive) from
#             the list of files to be scanned.
#         '=' signifies the entry is to be added, but if it is
#             a directory, then all its contents are pruned
#             (useful for /tmp).
#
# where: 'entry' is the absolute pathname of a file or a directory
#
# where 'attributes-flags' are in the format:
#        [template][ [+|-][pinugsam...] ... ]
#
#       - :  ignore the following atributes
#       + :  do not ignore the following attributes
#
#       p :  permission and file mode bits      a: access timestamp
```

```
#       i :  inode number                       m: modification timestamp
#       n :  number of links                    c: inode creation timestamp
#       u :  user id of owner                 md5: MD5 signature
#       g :  group id of owner              tiger: tiger signature*
#       s :  size of file                  rmd160: RMD160 signature*
#       b :  number of blocks (Linux)       sha1: SHA1 signature
#       S :  check for growing size
#
#       *) not implemented, - ignored
#
# Ex:   The following entry will scan all the files in /etc, and report
#       any changes in mode bits, inode number, reference count, uid,
#       gid, modification and creation timestamp, and the signatures.
#       However, it will ignore any changes in the access timestamp.
#
#       /etc    +p+i+n+u+g+s+m+md5+tiger+rmd160+sha1-a
#
# Note! '-' (minus) has presedence over '+' (plus), so the rule +p-p gives -p!
# Note! You MUST have assign +/- (plus/minus) for attributes! If not, the
#        attribute will be ignored!
#
# The following templates have been pre-defined to make these long ignore
# mask descriptions unecessary.
#
# Templates:
#  R         : [R]ead-only         (+p+i+n+u+g+s+b+m+md5+tiger+rmd160+sha1-a)
#  L         : [L]og file          (+p+i+n+u+g-s-b-a-m-md5-tiger-rmd160-sha1)
#  N         : ignore [N]othing    (+p+i+n+u+s+b+g+s+a+m+c+md5+tiger+rmd160+sha1)
#  E         : ignore [E]verything (-p-i-n-u-s-b-g-s-a-m-c-md5-tiger-rmd160-sha1)
#  > or Logs : growing logfile     (+p+u+g+i+n+S)
#  Binlib    : binary files        (+p+i+n+u+g+s+b+m+c+md5+sha1)
#  ConfFiles : config files        (+p+i+n+u+g+s+b+m+c+md5+sha1)
#  Devices   : device files /dev   (+p+i+n+u+g+s+b+c)
#  Databases : databases           (+p+n+u+g)
#  StaticDir : static directories  (+p+i+n+u+g)
#  ManPages  : manual pages        (+p+i+n+u+g+s+b+m+c+md5+sha1)
#
# You can use templates with modifiers, like:
#       Ex:  /etc/lp    E+u+g
#
#       Example configuration file:
#               /etc            R       # all system files
#               !/etc/lp        R       # ...but not those logs
#               =/tmp           N       # just the directory, not its files
#
# Note the difference between pruning (via '!') and ignoring everything
# (via 'E' template):  Ignoring everything in a directory still monitors
# for added and deleted files.  Pruning a directory will prevent Pide
# from even looking in the specified directory.
#
# Running slowly?  Modify the entries to ignore one of the signatures (md5
# or sha1) when this computationally-exorbitant protection is a paranoia
# setting only :^)
#

# Where is our database file?
database=file:/var/db/pide.db

# New databases (--init)
database_out=file:/var/db/pide.db.new

# Change this to 'no', or remove it to not gzip output
# (only useful on systems with few CPU cycles to spare)
gzip_dbout=yes
```

```
# Encrypt the database using AES with 256bit key?
# This require the blackbox module (and the Python
# Cryptographic Toolkit)
#encrypt_db=no

# Password to be used when encrypting/decrypting database
# NB! This should NOT be put here but given on the command line!
# You should have a *very* good reason for putting this here!
#password=secret

# Length of summary when doing --check or --update
#   short  = one line telling what's changed
#   normal = same as above and a list of changed files
#   detail = same as above and a detailed list of change
summary=normal

# Whether to warn about dead symlinks or not (default).
warn_dead_symlinks=no

# Custom rules/templates may go here
# NB!! '-' (minus) has presedence over '+' (plus), so the rule +p-p gives -p!
#Binlib = +p+i+n+u+g+s+b+m+c+md5+sha1

# You may specify environment variables to be exported when running here.
# It's normally only useful when used by the cron script.
# Define variable VAR to value val
# @@define VAR val
@@define MAILTO root@localhost
@@define LINES 1000

# You may undefine enviroment variables with
# @@undef VAR

# Next decide what directories/files you want in the database
#  - You may do globbing here; /bin/file*
#  - You may also use '$' to define end of file/dir (ex.: '/tmp$'), but
#    is not necessary, since python's glob does the job fine without -
#    so it's stripped. It's supported for compatibility with
#    aide/tripwire only.

# Homes
=/                L      # First, root's traditional home
/root             R      # Most likely root's  home
!/root/.bash_history
=/home            L      # Holding the users homedir

# Log files
=/var/log         StaticDir
/var/log          Logs

# Kernel, system map etc. - files that are used by the boot loader.
/boot             Binlib

# System configuration files
/etc              R
/usr/local/etc    R

# Binaries
/bin              Binlib
/sbin             Binlib
/usr/bin          Binlib
/usr/sbin         Binlib
/usr/local/bin    Binlib
```

```
/usr/local/sbin    Binlib
/usr/games         Binlib

# Libraries
/lib               Binlib
/usr/lib           Binlib
/usr/local/lib     Binlib

# Databases
/var/db            Databases

# Test only the directory when dealing with /proc and /tmp
=/proc             StaticDir
=/tmp              StaticDir

# You can look through these examples to get further ideas

# manpages can be trojaned, especially depending on *roff implementation
#/usr/man          ManPages
#/usr/share/man    ManPages
#/usr/local/man    ManPages

# docs
#/usr/doc          ManPages
#/usr/share/doc    ManPages

# check users' home directories
#/home             Binlib

# Devices - be careful to exclude terminals, sound devices, ++
#/dev              Devices

# check sources for modifications
#/usr/src          L
#/usr/local/src    L

# Check headers for same
#/usr/include       L
#/usr/local/include L

"""

########################## _error
def _error(err):
    """Exit if running standalone, else raise an exception
    """

    if __name__ == '__main__':
        print "%s: %s" % (os.path.basename(sys.argv[0]), str(err))
        print "Try `%s --help' for more information." % os.path.basename(sys.argv[0])
        sys.exit(1)
    else:
        raise Exception, str(err)

########################## _usage
def _usage():
    """Print usage of program
    """

    print """usage: %s [OPTIONS]
%s (Python Intrusion Detection Environment) is an intrusion
detection system for checking the integrity of files. Basically a
python implementation of Tripwire/AIDE.
```

```
Mandatory arguments to long options are mandatory for short options too.
  -i, --init         Initialize the database.
  -C, --check        Checks the database for inconsistencies. This is
                     the default option.
  -u, --update       Checks the database and updates the database
                     non-interactively.
  -c, --config=FILE  Read config options from FILE
  -d, --default      View an example config file. Do a '%s -d > pide.conf'
                     to pipe this to a config file.
  -e, --encrypt      Encrypt/decrypt the database when doing init or check.
  -p, --password=PASS  The password to be used when encrypting or
                     decrypting database.
  -v, --verbose      Level of debug messages.
  -h, --help         Display this help and exit.

Report bugs to lars [at] gnist org""" % (os.path.basename(sys.argv[0]), \
                                         os.path.basename(sys.argv[0]), \
                                         os.path.basename(sys.argv[0]))
########################## main
if __name__ == '__main__':
    """Main loop
    """

    # version control
    version = string.split(string.split(sys.version)[0][:3], '.')
    if map(int, version) < [2, 3]:
        _error('You need Python ver 2.3 or higher to run!')

    try:
        # opts = arguments recognized,
        # args = arguments NOT recognized (leftovers)
        opts, args = getopt.getopt(sys.argv[1:], "iCuhvc:e:p:d", \
                                   ["init", "check", "update", "help", "verbose", \
                                    "config=", "encrypt", "password=", "default"])
    except getopt.GetoptError:
        # print help information and exit:
        _error('illegal option(s) -- %s' % str(sys.argv[1:]))

    # try this as default pide.conf file, if none given
    rcfile = '/etc/pide.conf'

    # default variables
    verbose = 0; crypt = 0; password = ''

    # get config file before init
    for o, a in opts:
        if o == "-c" or o == "--config": # read config from this file
            try:   # check for readability later
                rcfile = str(a)
            except:
                _error("invalid config file")
        if o == '-v' or o == "--verbose":
            verbose = 1
        if o == '-e' or o == '--encrypt':
            if not have_blackbox:
                _error("ERROR: Can't find the blacbox module!")
            else:
                crypt = 1
        if o == '-p' or o == '--password':
            if not have_blackbox:
                _error("ERROR: Can't find the blackbox module!")
            else:
                try:
                    password = str(a)
```

```
                    crypt = 1
                except:
                    _error('ERROR: Invalid password')

    for o, a in opts:
        if o == "-h" or o == "--help":   # display help and exit
            _usage()
            sys.exit(0)
        if o == "-d" or o == "--default": # display example config file
            printconf()
            sys.exit(0)
        if o == "-i" or o == "--init":   # initialize the database
            integrityCheck(init=1, rcfile=rcfile, verbose=verbose, \
                           password=password, crypt=crypt)
            sys.exit(0)
        if o == "-C" or o == "--check":  # check for inconsistecies
            integrityCheck(check=1, rcfile=rcfile, verbose=verbose, \
                           password=password, crypt=crypt)
            sys.exit(0)
        if o == "-u" or o == "--update": # update the database
            integrityCheck(update=1, rcfile=rcfile, verbose=verbose, \
                           password=password, crypt=crypt)
            sys.exit(0)

    # if there is no options, do default (--check)
    integrityCheck(check=1, rcfile=rcfile, password=password, crypt=crypt)

# end
```

Appendix C

Default configuration file for PIDE

pide.conf

```
#
# pide.conf based on Tripwire's tw.config
#
# NOTE! the tiger and rmd160 checksum is NOT implemented! So it
# it will be ignored (but md5 and sha1 should do the job).
#
# This file contains a list of files and directories that this
# script will scan. Information collected from these files will be
# stored in a database file.
#
# Format: [!|=] entry [attributes-flags]
#
# where:  '!' signifies the entry is to be pruned (inclusive) from
#             the list of files to be scanned.
#         '=' signifies the entry is to be added, but if it is
#             a directory, then all its contents are pruned
#             (useful for /tmp).
#
# where: 'entry' is the absolute pathname of a file or a directory
#
# where 'attributes-flags' are in the format:
#        [template][ [+|-][pinugsam...] ... ]
#
#       - :  ignore the following atributes
#       + :  do not ignore the following attributes
#
#       p :  permission and file mode bits      a: access timestamp
#       i :  inode number                       m: modification timestamp
#       n :  number of links                    c: inode creation timestamp
#       u :  user id of owner                 md5: MD5 signature
#       g :  group id of owner              tiger: tiger signature*
#       s :  size of file                 rmd160: RMD160 signature*
#       b :  number of blocks (Linux)       sha1: SHA1 signature
#       S :  check for growing size
#
#       *) not implemented, - ignored
#
# Ex:   The following entry will scan all the files in /etc, and report
#       any changes in mode bits, inode number, reference count, uid,
#       gid, modification and creation timestamp, and the signatures.
#       However, it will ignore any changes in the access timestamp.
#
#       /etc    +p+i+n+u+g+s+m+md5+tiger+rmd160+sha1-a
#
# Note! '-' (minus) has presedence over '+' (plus), so the rule +p-p gives -p!
# Note! You MUST have assign +/- (plus/minus) for attributes! If not, the
#        attribute will be ignored!
#
```

```
# The following templates have been pre-defined to make these long ignore
# mask descriptions unecessary.
#
# Templates:
#  R         : [R]ead-only         (+p+i+n+u+g+s+b+m+md5+tiger+rmd160+sha1-a)
#  L         : [L]og file          (+p+i+n+u+g-s-b-a-m-md5-tiger-rmd160-sha1)
#  N         : ignore [N]othing    (+p+i+n+u+s+b+g+s+a+m+c+md5+tiger+rmd160+sha1)
#  E         : ignore [E]verything (-p-i-n-u-s-b-g-s-a-m-c-md5-tiger-rmd160-sha1)
#  > or Logs : growing logfile     (+p+u+g+i+n+S)
#  Binlib    : binary files        (+p+i+n+u+g+s+b+m+c+md5+sha1)
#  ConfFiles : config files        (+p+i+n+u+g+s+b+m+c+md5+sha1)
#  Devices   : device files /dev   (+p+i+n+u+g+s+b+c)
#  Databases : databases           (+p+n+u+g)
#  StaticDir : static directories  (+p+i+n+u+g)
#  ManPages  : manual pages        (+p+i+n+u+g+s+b+m+c+md5+sha1)
#
# You can use templates with modifiers, like:
#       Ex:  /etc/lp    E+u+g
#
#       Example configuration file:
#               /etc            R       # all system files
#               !/etc/lp        R       # ...but not those logs
#               =/tmp           N       # just the directory, not its files
#
# Note the difference between pruning (via '!') and ignoring everything
# (via 'E' template):  Ignoring everything in a directory still monitors
# for added and deleted files.  Pruning a directory will prevent Pide
# from even looking in the specified directory.
#
# Running slowly?  Modify the entries to ignore one of the signatures (md5
# or sha1) when this computationally-exorbitant protection is a paranoia
# setting only :^)
#

# Where is our database file?
database=file:/var/db/pide.db

# New databases (--init)
database_out=file:/var/db/pide.db.new

# Change this to 'no', or remove it to not gzip output
# (only useful on systems with few CPU cycles to spare)
gzip_dbout=yes

# Encrypt the database using AES with 256bit key?
# This require the blackbox module (and the Python
# Cryptographic Toolkit)
#encrypt_db=no

# Password to be used when encrypting/decrypting database
# NB! This should NOT be put here but given on the command line!
# You should have a *very* good reason for putting this here!
#password=secret

# Length of summary when doing --check or --update
#   short  = one line telling what's changed
#   normal = same as above and a list of changed files
#   detail = same as above and a detailed list of change
summary=normal

# Whether to warn about dead symlinks or not (default).
warn_dead_symlinks=no

# Custom rules/templates may go here
```

```
# NB!! '-' (minus) has presedence over '+' (plus), so the rule +p-p gives -p!
#Binlib = +p+i+n+u+g+s+b+m+c+md5+sha1

# You may specify environment variables to be exported when running here.
# It's normally only useful when used by the cron script.
# Define variable VAR to value val
# @@define VAR val
@@define MAILTO root@localhost
@@define LINES 1000

# You may undefine enviroment variables with
# @@undef VAR

# Next decide what directories/files you want in the database
#  - You may do globbing here; /bin/file*
#  - You may also use '$' to define end of file/dir (ex.: '/tmp$'), but
#    is not necessary, since python's glob does the job fine without -
#    so it's stripped. It's supported for compatibility with
#    aide/tripwire only.

# Homes
=/                L      # First, root's traditional home
/root             R      # Most likely root's  home
!/root/.bash_history
=/home            L      # Holding the users homedir

# Log files
=/var/log         StaticDir
/var/log          Logs

# Kernel, system map etc. - files that are used by the boot loader.
/boot             Binlib

# System configuration files
/etc              R
/usr/local/etc    R

# Binaries
/bin              Binlib
/sbin             Binlib
/usr/bin          Binlib
/usr/sbin         Binlib
/usr/local/bin    Binlib
/usr/local/sbin   Binlib
/usr/games        Binlib

# Libraries
/lib              Binlib
/usr/lib          Binlib
/usr/local/lib    Binlib

# Databases
/var/db           Databases

# Test only the directory when dealing with /proc and /tmp
=/proc            StaticDir
=/tmp             StaticDir

# You can look through these examples to get further ideas

# manpages can be trojaned, especially depending on *roff implementation
#/usr/man          ManPages
#/usr/share/man    ManPages
#/usr/local/man    ManPages
```

```
# docs
#/usr/doc          ManPages
#/usr/share/doc    ManPages

# check users' home directories
#/home             Binlib

# Devices - be careful to exclude terminals, sound devices, ++
#/dev              Devices

# check sources for modifications
#/usr/src          L
#/usr/local/src    L

# Check headers for same
#/usr/include       L
#/usr/local/include L

```

Appendix D

Source code blackbox.py

blackbox.py

```
#!/usr/bin/env python
# -*- coding: iso-8859-1 -*-
"""
blackbox.py

What is it?

  - an abstraction layer to ease the use of the 'Python Cryptography Toolkit'
  - implements padding/stripping of messages to nearest block size.
    Required by some the cipher block mode used (CBC)
  - Implements the PKCS#5 v2.0: Password-Based Cryptography Standard
    from RSA Laboratories. RFC2898 http://www.rfc-editor.org/rfc/rfc2898.txt

When running as a standalone program, it can be used to encrypt/decrypt files.

There are two classes, namly:

  Reactor - handles encryption/decryption, padding/stripping, generation of IV,
            checking for correct password, generation of head
  PBKDFv2 - handles the generation of key from a given password


Copyright (C) 2004 - Lars Strand <lars strand at gnist org>

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA

http://www.gnu.org/copyleft/gpl.html


TODO:
 DONE --> save 42bits of sha1 of password at start of message
 DONE --> save IV at start of message
 DONE --> take commando options
 DONE --> verbose setting
 DONE --> change algoritm
```

```
"""

import sha, getopt, sys, os, os.path, getpass, md5, time, random
import struct, base64, string, math, hmac # RFC2104

# Package definitions.
__program__   = 'Blackbox'
__version__   = '0.6b'
__date__      = '2004/28/09'
__author__    = 'Lars Strand <lars at unik no>'
__licence__   = 'GPL'
__copyright__ = 'Copyright (C) 2004 Lars Strand'

# code to use when padding files
STRING = 0x01
FILE = 0x02

################ Reactor
class Reactor:
    """Reactor - a class to encrypt/decrypt files/messages using a
    given symmetric encryption algoritm. Using the 'Python Cryptography
    Toolkit' from http://www.amk.ca/python/code/crypto.html"""

    def __init__(self, password, algorithm):
        """Initialize the class with the the encrypt key, mode
        block size and cipher to be used"""

        from Crypto.Cipher import DES3, Blowfish, AES

        #self.mode = AES.MODE_CBC # ECB #CBC # CFB              # cipher mode to use
        self.block_size = AES.block_size      # cipher block mode of operation
        self.keysize = AES.key_size           # lenght of key? Zero if variable..
        if self.keysize == 0: self.keysize = 256  # ..so we set keysize
        self.keysizebyte = (self.keysize/8)   # to get key in byte
        self.cipher = AES.new                 # new AES object
        # not necessary when decrypting, but is prettier (when generating keys etc.)
        self.IV = self.generateIV()
        self.password = password              # is hashed before stored
        self.passHashSize = 4                 # nr of bytes the stored hashed password should c

        # What algorithm to use? AES is the default
        if algorithm == 'DES3':
            self.mode = DES3.MODE_CBC
        elif algorithm == 'Blowfish':
            self.mode = Blowfish.MODE_CBC
        elif algorithm == 'AES' or algorithm == '':
            self.mode = AES.MODE_CBC
        else:
            print "WARNING! Unknow cipher algorithm %s!" % algorithm
            print "Using fallback algorithm AES"
            self.mode = AES.MODE_CBC

    ### checkPassword
    def checkPassword(self, ciphertext):
        """Gets a plaintext password, and a ciphermessage. At the
        start of the ciphermessage is part of a cipher key. The
        salt used to hash the password is the IV. The length of the
        IV is block_size.

        1. Fecth the password-part and IV

        2. Compute a hash using the password and IV

        3. Compare the two
```

```
        4. Return 1 if correct, else return 0
        """

        # fetch password and IV
        password = ciphertext[:self.passHashSize]
        IV = ciphertext[self.passHashSize:self.passHashSize+self.block_size]

        # compute hash
        hash = sha.new(self.password + str(IV))

        # the same?
        if str(hash.hexdigest()[:self.passHashSize]) == str(password):
            return 1
        else:
            return 0

    ### generates a IV
    def generateIV(self):
        """Produces a initialization vector (IV) to be used when
        dealing with Cipher Feedback Mode (CFB) mode.

        Input: Blocksize

        Generate: System time and a random selected number is SHA-1
        hashed (a radom number to make it a little harder to find
        out when the IV was generated). A plain random number does
        not contain letters, so a hash i preferred as a pseudorandom
        generator.

        Output: Returns the number of chars as blocksize.
        """

        random.seed()
        hash = sha.new()               # create a new hash object
        hash.update(str(time.time())+str(random.random()))  # feed the hash
        iv = hash.hexdigest()          # create the hash

        # problem: what if there is larger block size than hash?
        # IV = blocksize = hash!
        # solution - just add the same hash until long enough
        while self.block_size > len(iv):
            iv += iv

        return iv[:self.block_size]  # return IV equal block size

    ### generateHead
    def generateHead(self):
        """At the start of the ciphermessage is part of a hashed password
        and the initial vector (IV). The message will look like this

        <password><IV><ciphermessage>

        where

        <password> - is the password and IV (used as salt) hashed using SHA1.
        The sha1 produces 160bits (40 bytes) - but only 42 bits (6 bytes) are
        stored. This way an potensial attacker will only be able to operate
        on a small portion of the hash (when f.ex. brute force) - but when
        checked against a password, this will only fails (when typed correct
        password) every 2^-42 times.

        <IV> - the initialization vector

        <ciphermessage> - the encrypted message
```

```
        This functions generates the <password><IV> string = HEAD
        """

        if len(self.password) <= 7:
            raise "ERROR: password must be at least 8 characters long!"

        hash = sha.new(self.password+self.IV)

        return str(hash.hexdigest()[:self.passHashSize]) + str(self.IV)


    ### Pad plaintext to blocksize
    def pad(self, msg, type):
        """Takes a message and a blocksize - returns the
        message padded to nearest blocksize. The rfc2315 gives some
        small hints on how to do that:

        http://www.ietf.org/rfc/rfc2315.txt

        Taken from section '10.3 Content-encryption process':

        '2.   Some content-encryption algorithms assume the
        input length is a multiple of k octets, where k > 1, and
        let the application define a method for handling inputs
        whose lengths are not a multiple of k octets. For such
        algorithms, the method shall be to pad the input at the
        trailing end with k - (l mod k) octets all having value k -
        (l mod k), where l is the length of the input. In other
        words, the input is padded at the trailing end with one of
        the following strings:

        01 -- if l mod k = k-1
        02 02 -- if l mod k = k-2
        .
        .
        .
        k k ... k k -- if l mod k = 0

        The padding can be removed unambiguously since all input is
        padded and no padding string is a suffix of another. This
        padding method is well-defined if and only if k < 256;
        methods for larger k are an open issue for further study.'

        The data, with padding will look like:

        <code><size><string/data><padding>
        """

        # If type is 'string', pad the message accordingly:
        # The string is padded:
        # <type-byte><length><thestring><padding>
        if type == STRING:

            # input the type of message in start of file
            r = chr(STRING)+'0'+msg                     # gives: <type-byte><length><string>
            #bitlength = len(r) * 8                      # number of bits in msg
            remainder = float(len(r)) % self.block_size  # left-overs?

            # if there is a remainder - we must pad the message
            if remainder:
                padlength  = self.block_size - remainder # number of bytes to pad
                #bytepadlength = int(bitpadlength)/8     # number of bytes (char) to pad

                # Can't pad more than 255 chars because chr(x) can't
```

```
            # take higher x than 255 (ASCII values).
            # ==> Max blocksize: 256 * 8 = 2048!
            if padlength > (256*8): raise "Can't pad more than 2048 bits! Blocksize is to l

            # Okay, do padding!
            padding = ""
            for i in range(int(padlength)):       # pad number of chars as "leftover" chars

                padding += chr(int(padlength))    # paddmsg to add to origingal message
                # The padded msg:
                # - first apply the type of padded msg = 01 = STRING
                # - then insert number of padded bytes - this is max 256 - but that is alsc
                #   the limit for block size (256*8=2048bits)
                #   Why size here? What if a long string contains all lettern 'N' = ASCII v
                #   First check last byte - what number is that? And then count backwards 7
                #   and if that also is the letter 'N' (78) - remove them all - but actuall
                #   string didn't need to be padded! - So *must* have a size!
                # - the original msg
                # - then the padding
            return chr(STRING)+chr(padlength)+msg+bytepadding  # return padded message

        else: # no need to pad message
            return chr(STRING)+chr(0)+msg

    # We're dealing with a file
    elif type == FILE:

        # If there is a file
        if os.path.isfile(msg):

            # get the filesize
            try:
                filemode = os.stat(msg)
            except: raise "ERROR! Unable to stat file!"

            # add 2 because of <type> and <size> (bytes)
            bytesize = filemode.st_size+1+1

            # how much to pad?
            remainder = float(bytesize) % self.block_size
            padlength = self.block_size - remainder

            # open file
            try:
                filehandle = open(msg, 'rb')
            except:
                raise "ERROR! Unable to open file!"

            # read file
            filedata = filehandle.read(bytesize)

            if remainder:
                # Can't pad more than 255 chars because chr(x) can't
                # take higher x than 255 (ASCII values).
                # ==> Max blocksize: 256 * 8 = 2048!
                if padlength > (256*8): raise "Can't pad more than 2048 bits! Blocksize is

                # construct padding
                padding = ""
                for i in range(int(padlength)):  # pad number of chars as "leftover" chars
                    padding += chr(int(padlength))     # paddmsg to add to origingal messag

                # return <code><size>+file+padding
                return chr(FILE)+chr(int(padlength))+filedata+padding
```

```
            # no need to pad file
            else:
                return chr(FILE)+chr(int(padlength))+filedata
        else:
            raise "ERROR! %s is not a file!" % msg

    else: raise "ERROR! Unknown object type to pad! Must be 'string' or 'file'!"

### encrypts a given plaintext
def encrypt(self, plaintext, type):
    """Takes a plaintext an type as input. Returns ciphertext.

    1. Generate an encryption key pased on the password - according
       to RFC2898. The PBKDFv2 class does the work.
    2. Pad the message to nearest block, using pad(). Message format
       is then:
        <type><size><plaintext><padding>
    3. Generate the head using generateHead(). The head consist og
       a small protion of a hashed password and the IV in plaintext.
    4. Encrypt the message and return <head+ciphertext>"""

    # 1. make a encryption key based on password
    gkey = PBKDFv2()
    self.key = gkey.makeKey(self.password, self.IV, 1000, self.keysizebyte)

    # 2. pad message to nearest block size
    padded = self.pad(plaintext, type)

    # 3. generate the head
    head = self.generateHead()

    # 4. encrypt and return head+ciphermessage
    return head + self.cipher(self.key, self.mode, self.IV).encrypt(padded)

### decrypt a given ciphertext
def decrypt(self, ciphertext):
    """Takes a given ciphertext, decrypt and returns plaintext.

    1. Fetch the IV at the start of ciphertext.
    2. Generate an encryption key pased on the password - according
       to RFC2898. The PBKDFv2 class does the work.
    3. Decrypt the message.
    4. Fecth the type of message (FILE? STRING?) and size
    5. Strip the message of padding and return the plaintext
    """

    # 1. the IV is located as a header of the ciphertext
    # <part-of-password><IV><ciphermessage>
    self.IV = ciphertext[self.passHashSize:self.passHashSize+self.block_size]

    # 2. make a encryption key based on password
    gkey = PBKDFv2()
    self.key = gkey.makeKey(self.password, self.IV, 1000, self.keysizebyte)

    # 3. decrypt the message
    plaintext = self.cipher(self.key, self.mode, self.IV).decrypt(ciphertext[self.passHashS

    # 4. get the type and size of message
    typ = plaintext[0:1]
    size = ord(plaintext[1:2])

    # 5. strip and return the plaintext
    return plaintext[2:-size]
```

```
################ PBKDFv2
class PBKDFv2:
    """Implements the PKCS#5 v2.0: Password-Based Cryptography Standard
    from RSA Laboratories. RFC2898

    http://www.rfc-editor.org/rfc/rfc2898.txt
    """

    ################ init
    def __init__(self):

        # length of pseudorandom function: 20 for SHA-1, 16 for MD5
        self.hLen = 20

    ################ makeKey
    def makeKey(self, P, S, c, dkLen):
        """Options: PRF   underlying pseudorandom function (hLen denotes the length
                    of the pseudorandom function output - 16 for MD5 or 20 for SHA-1)
                    Using standard module module hmac to do this.

           Input:   P        password, an octet string
                    S        salt, an octet string
                    c        iteration count, a positive integer (>1000)
                    dkLen    intended length on octets of the derived key, a positive integer,
                             at most (2^32 - 1) * hLen

           Output   DK    derived key, a dkLen-octet string
           """

        # do some sanity checks
        try:
            str(P); str(S); int(c); float(dkLen); int(c)
        except:
            print "P = %s, S = %s, c = %s, dkLen = %s:" % (P, S, c, dkLen)
            raise "ERROR! Input is not correct!"

        if len(P) < 8: raise "ERROR! Password is to short! Min 8 char!"
        if len(S) < 8: raise "ERROR! Salt is to short! Min 8 char!"
        if c < 1000: raise "ERROR! Number of iteration 'c', must be at least 1000!"

        # Step 1: if dkLen is larger than maximum possible key - exit
        if dkLen > ((2^32 - 1) * self.hLen):
            maxlength = (2^32 - 1) * self.hLen
            raise "ERROR! Key is to large! Maxlength is", str(maxlength)

        # Step 2:
        # Let l be the number of hLen-octet blocks in the derived key, rounding up
        # and let r be the number of octets in the last block
        l = math.ceil(dkLen / float(self.hLen))
        #if (dkLen % float(self.hLen)): l = int(l) + 1 # round up if necessary
        r = dkLen - (l - 1) * self.hLen

        # Step 3:
        # For each block of the derived key, apply the function F to the
        # password P, the salt S, the iteration count c, and the block index
        # to compute the block
        T = ""
        for blockindex in range(int(l)):
            T += self.F(P, S, c, blockindex)
        # Step 4 - extract the first dkLen octet to produce a derived key DK
        DK = T[:dkLen]

        # Step 5 - return the derived key DK
        return DK
```

```
    ################ F
    def F(self, P, S, c, i):
        """For each block of the derived key, apply this function.

        Notation:
        ||   = concatenation operator
        PRF  = Underlying pseudorandom function

        The function F is defined as the exclusive-or sum of the first c
        iterates if the underlying pseudorandom function PRF applied to
        the password P and the concatenation of the salt S and the block
        index i:

        F(P, S, c, i) = U1 XOR U2 XOR ... XOR Uc

        where

        U1 = PRF(P, S || INT(i)),
        U2 = PRF(P, U1)
        ...
        Uc = PRF(P, Uc-1)
        """

        # The pseudorandom function, PRF, used is HMAC-SHA1 (rfc2104)
        iteration = 1

        # the first iteration; P is the key, and a concatination of
        # S and blocknumber is the message
        PRF = hmac.new(P, str(S)+str(i), sha)
        U = PRF.digest() # the first iteration

        while iteration < c:                  # loop through all iterations
            PRF = hmac.new(P, U, sha)         # a new iteration
            U = self._xor(U, PRF.digest())    # XOR this new iteration with the old one
            iteration += 1
        return U

    ################ xor
    def _xor(self, a, b):
        """Performs XOR on two strings a and b"""

        import string

        if len(a) != len(b):
            raise "ERROR: Strings are of different size! %s %s" % (len(a), len(b))

        result = ""
        for x in range(len(a)):
            # first convert each character to an ASCII code (integer)
            # using ord, do XOR and then convert back to ASCII character (chr)
            result += chr(ord(a[x]) ^ ord(b[x]))

        return result

############################ usage
def usage():
    """Print usage of program"""
    print """usage: %s [OPTIONS] file1 file2
%s is a wrapper for the 'Python Cryptography Toolkit' from
http://www.amk.ca/python/code/crypto.html
You may use this script to encrypt/decrypt files. It's also suitable
as a Python module.

Mandatory arguments to long options are mandatory for short options too.
```

```
    -e, --encrypt          Encrypt the input data (default)
    -d, --decrypt          Decrypt input data
    -a, --algortihm=CIPHER Cipher algorithm to use. Supported ciphers are:
                           DES3, Blowfish and AES (default)
                           NB! Do remember to use the same algorithm on both
                           encrypt and decrypt!
    -p, --password=PASS    The password to derive the key from
    -o, --out=FILE         The output of the encrypted/decrypted data. Default:
                           append '.crypt' at end of each encrypted file and
                           '.decrypted' of each decrypted file
    -v, --verbose          Show verbose options
    -h, --help             Display this help and exit.

  Report bugs to lars [at] gnist org""" % (os.path.basename(sys.argv[0]), os.path.basename(sys.ar

  ################ end class PBKDFv2

  if __name__ == '__main__':

      import glob

      # do not proceed if can't find crypto module
      try:
          from Crypto.Cipher import AES
      except:
          print "ERROR: You don't have the crypto modules installed!"
          print "Download and install the 'Python Cryptography Toolkit'"
          print "from: http://www.amk.ca/python/code/crypto.html"
          sys.exit(1)

      decrypt = 0; encrypt = 1; password = ""; verbose = 0; algorithm = ""
      data = []; encout = ''; decout = ''

      # read variables from the command line, one by one:
      while len(sys.argv) >= 2:
          option = sys.argv[1];       del sys.argv[1]
          if option == '-e' or option == '--encrypt':
              encrypt = 1
          elif option == '-d' or option == '--decrypt':
              decrypt = 1; encrypt = 0
          elif option == '-a' or option == '--algorithm':
              try:
                  algorithm = str(sys.argv[1])
              except:
                  print "ERROR! You must give value to this option!"
                  sys.exit(1)
          elif option == '-p' or option == '--password':
              try:
                  password = str(sys.argv[1])
              except:
                  print "ERROR! You must give value to this option!"
                  sys.exit(1)
          elif option == '-o' or option == '--out':
              try:
                  encout = str(sys.argv[1])
                  decout = str(sys.argv[1])
              except:
                  print "ERROR! You must give value to this option!"
                  sys.exit(1)
          elif option == '-v' or option == '--verbose':
              verbose = 1
          elif option == '-h' or option == '--help':
              usage()
              sys.exit(0)
```

```
        else:
            for i in glob.glob(option):
                data.append(i)

    if verbose:
        for fi in data:
            print "Added file ==>", fi

    # exit if no valid filenames
    if len(data) == 0:
        print "ERROR: No valid file names given!"
        print "Try '%s --help' for more information." % os.path.basename(sys.argv[0])
        sys.exit(1)

    # Read password from user
    while len(password) <= 7:
        print "Password must be at least 8 char long!"
        password = getpass.getpass("Enter Password: ")

    # initiate the crypt class
    if verbose: print "Initializing cryptoclass..",
    crpt = Reactor(password, algorithm)
    if verbose: print "done"

    # decrypt or encrypt?
    if encrypt:
        if verbose: print "Doing encrypt"
        for f in data:
            if not encout: encout = f+'.encrypted'
            # encrypt the file
            if verbose: print "Encrypting file ==> %s", f
            cryptfile = crpt.encrypt(f, FILE)
            if verbose: print "done!"
            # trying to save the encrypted file
            print "Saving encrypted file ==> %s", encout
            try:
                ofile = open(encout, 'w')
                ofile.write(cryptfile)
                ofile.close()
            except:
                print "ERROR! Unable to write encrypted file! %s" % encout
    else: # we're doing decrypt
        if verbose: print "Doing encrypt"
        for f in data:
            if not decout: decout = f+'.decrypted'
            # Get the file's size and read it
            try:
                size = os.path.getsize(f)
                ifile = open(f, 'rb')
                cryptfile = ifile.read(size)
                ifile.close()
            except:
                print "ERROR: Unable to open %s!" % f
                continue
            # check for correct password
            if crpt.checkPassword(cryptfile):
                # decrypting the file
                if verbose: print "Decrypting file %s" % f
                decryptfile = crpt.decrypt(cryptfile)
                # saving the file
                print "Saving decrypted file ==> %s" % decout
                try:
                    ofile = open(decout, 'w')
                    ofile.write(decryptfile)
```

```
                    ofile.close()
630             except:
                    print "ERROR! Could not write crypto to file: %s" % decout
            else:
                print "ERROR! Wrong password!"

635 if verbose: print "all done"
```

Bibliography

[1] Bernard Aboba, Larry J. Blunk, John R. Vollbrecht, James Carlson, and Henrik Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748, Internet Engineering Task Force (IETF), June 2004. `http://www.ietf.org/rfc/rfc3748.txt`.

[2] Bernard Aboba and Dan Simon. PPP EAP TLS Authentication Protocol. RFC 2716, Internet Engineering Task Force (IETF), October 1999. `http://www.ietf.org/rfc/rfc2716.txt`.

[3] James P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, Fort Washington, Feb 1980.

[4] Steven Bellovin. The Security Flag in the IPv4 Header. RFC 3514, Internet Engineering Task Force (IETF), April 2003. `http://www.ietf.org/rfc/rfc3514.txt`.

[5] Steven M. Bellovin. Distributed Firewalls. *;login: magazine, special issue on security*, pages 37–39, Nov 1999.

[6] Steven M. Bellovin and William R. Cheswick. Network Firewalls. *IEEE Communications Magazine*, pages 50–57, Sep 1994.

[7] K. Biba. Integrity consideration for secure computer systems. Technical Report MTR-3153, MITRE Corporation, 1975.

[8] Matthew Bishop. *Computer Security: Art and Science*. Addison Wesley, Dec 2002.

[9] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, Internet Engineering Task Force (IETF), September 1999. `http://www.ietf.org/rfc/rfc2704.txt`.

[10] R. Braden, D. Borman, and C. Partridge. Computing the Internet checksum. RFC 1071, Internet Engineering Task Force (IETF), September 1988. `http://www.ietf.org/rfc/rfc1071.txt`.

[11] Canadian System Security Centre. *The Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)*, January 1993. Version 3.0e.

[12] Canadian System Security Centre. *Common Criteria for Information Technology Security Evaluation (CC)*, August 1999. Version 2.1.

[13] CERT Coordination Center. TCP SYN Flooding and IP Spoofing Attacks. Advisory CA-1996-21, CERT Coordination Center, November 1996. `http://www.cert.org/advisories/CA-1996-21.html`.

[14] William R. Cheswick, Steven M. Bellovin, and Aviel D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison Wesley, second edition, Mar 2003.

[15] D. Clark and D. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194, Apr 1987.

[16] Thomas Heide Clausen and Philippe Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, Internet Engineering Task Force (IETF), October 2003. `http://www.ietf.org/rfc/rfc3626.txt`.

[17] Commission of the European Communities. *Information Technology Security Evaluation Criteria (ITSEC)*, June 1991.

[18] daemon9, route, and infinity. Project Neptune. *Phrack*, Jul 1996. `http://www.phrack.org/show.php?p=48&a=13`.

[19] David E. Bell and Leonard J. LaPadula. Secure Computer Systems: A Mathematical Model. Technical Report ESD-TR-73-278, MTR-2547, Vol I, MITRE Corporation, Bedford, MA, Nov 1973.

[20] David E. Bell and Leonard J. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, MTR-2997 Rev. 1, MITRE Corporation, Bedford, MA, Mar 1976.

[21] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, Feb 1987.

[22] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, Internet Engineering Task Force (IETF), January 1999. `http://www.ietf.org/rfc/rfc2246.txt`.

[23] DOD. Dictionary of Military and Associated Terms. Joint Publication 1-02, US Department of Defence, Apr 2001. `http://www.dtic.mil/doctrine/jel/new_pubs/jp1_02.pdf`.

[24] Donald E. Eastlake and Paul E. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, Internet Engineering Task Force (IETF), September 2001. `http://www.ietf.org/rfc/rfc3174.txt`.

[25] John C. Klensin (ed.). Simple Mail Transfer Protocol. RFC 2821, Internet Engineering Task Force (IETF), April 2001. `http://www.ietf.org/rfc/rfc2821.txt`.

[26] Paul Ferguson and Daniel Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2267, Internet Engineering Task Force (IETF), January 1998. `http://www.ietf.org/rfc/rfc2267.txt`.

[27] Vince Fuller, Tony Li, Jessica (Jie Yun) Yu, and Kannan Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519, Internet Engineering Task Force (IETF), September 1993. `http://www.ietf.org/rfc/rfc1519.txt`.

[28] Dieter Gollmann. *Computer Security*. John Wiley and Sons Ltd, Jul 1998.

[29] Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn, and Robert Richardson. 2004 CSI/FBI Computer Crime and Securty Survey. Annual Report, Computer Security Institute, 2004. `http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2004.pdf`.

[30] Geir Hallingstad, Martin Gilje Jaatun, and Ronny Windvik. Firewall technology. DRAFT, 2000.

[31] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion Detection using Sequences of System Calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[32] Information Sciences Institute University of Southern California. Internet Protocol. RFC 791, Internet Engineering Task Force (IETF), September 1981. http://www.ietf.org/rfc/rfc791.txt.

[33] Information Sciences Institute University of Southern California. Transmission Control Protocol. RFC 793, Internet Engineering Task Force (IETF), September 1981. http://www.ietf.org/rfc/rfc793.txt.

[34] Institute of Electrical and Electronics Engineers. Local and metropolian area networks: Port-Based Network Access Control. IEEE Standard 802.1X-2001, IEEE Computer Society, October 2001.

[35] Institute of Electrical and Electronics Engineers. Medium Access Control (MAC) Security Enhancments. IEEE Standard 802.11i-2004, IEEE Computer Society, June 2004.

[36] International Telecommunication Union (ITU). Security Architecture For Open Systems Interconnection (OSI). X.800 Standard X.800, The International Telegraph and Telephone Consultative Comittee (CCITT), 1991.

[37] Sotris Ioannidis, Angelos D. Keromytis, Steven M. Bellovin, and Jonathan M. Smith. Implementing a Distributed Firewall. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199, Sep 2000.

[38] J. Postel. Internet Control Message Protocol. RFC 792, Internet Engineering Task Force (IETF), September 1981. http://www.ietf.org/rfc/rfc792.txt.

[39] Joan Daemen and Vincent Rijmen. *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology, November 2001. FIPS PUB 197.

[40] Christopher A. Jones and Fred L. Drake Jr. *Python and XML*. O'Reilly, 2002.

[41] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631, Internet Engineering Task Force (IETF), May 1994. http://www.ietf.org/rfc/rfc1631.txt.

[42] David Kahn. *The Codebreakers*. Scribner, second edition, 1996.

[43] B. Kaliski. PKCS #5: Password-Based Cryptography Specification. Version 2.0. RFC 2898, Internet Engineering Task Force (IETF), September 2000. http://www.ietf.org/rfc/rfc2898.txt.

[44] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, Internet Engineering Task Force (IETF), February 1997. http://www.ietf.org/rfc/rfc2104.txt.

[45] Lars Strand. *Linux Optimized Link State Routing Protocol (OLSR) IPv6 HOWTO*, August 2003. http://www.tldp.org/HOWTO/OLSR-IPv6-HOWTO/.

[46] Lars Strand. *802.1X Port Based Authentication HOWTO*, August 2004. http://tldp.org/HOWTO/8021X-HOWTO/.

[47] S. Lipner. Non-Discretionary Controls for Commercial Applications. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pages 2–10, Apr 1982.

[48] Teresa F. Lunt and R. Jagannathan. A Prototype Real-Time Intrusion-Detection Expert-System. In *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on*, pages 59–66, 1988.

[49] Jeffry C. Mogul, Richard F. Rashid, and Michael J. Accetta. The packer filter: an efficient mechanism for user-level network code. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 39–51, 1987.

[50] National Institute of Standards and Technology. *Data Encryption Standard (DES)*. U.S. Department of Commerce, January 1977. FIPS PUB 46-3 (Reaffirmed Oct 25 1999).

[51] National Institute of Standards and Technology. *Computer Data Authentication*. U.S. Department of Commerce, May 1985. FIPS PUB 113.

[52] Basavaraj Patil, Phil Roberts, and Charles E. Perkins. IP Mobility Support for IPv4. RFC 3344, Internet Engineering Task Force (IETF), August 2002. `http://www.ietf.org/rfc/rfc3344.txt`.

[53] P.A. Porras. STAT – A State Transition Analysis Tool for Intrusion Detection. Master's thesis, Computer Science Department, University of California, Santa Barbara, June 1992.

[54] J. Postel and J. Reynolds. File Transfer Protocol (FTP). RFC 959, Internet Engineering Task Force (IETF), October 1985. `http://www.ietf.org/rfc/rfc959.txt`.

[55] Yakov Rekhter and Tony Li. An Architecture for IP Address Allocation with CIDR. RFC 1518, Internet Engineering Task Force (IETF), September 1993. `http://www.ietf.org/rfc/rfc1518.txt`.

[56] Yakov Rekhter and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, Internet Engineering Task Force (IETF), March 1995. `http://www.ietf.org/rfc/rfc1771.txt`.

[57] Ron Rivest. The MD5 Message-Digest Algorithm. RFC 1321, Internet Engineering Task Force (IETF), April 1992. `http://www.ietf.org/rfc/rfc1321.txt`.

[58] Bruce Schneier. *Applied Crypthography*. Wiley, second edition, 1996.

[59] Bruce Schneier. A Hardware DES Cracker. *Crypto-Gram Newsletter*, Aug 1998. `http://www.schneier.com/crypto-gram-9808.html#descracker`.

[60] Bruce Schneier. News. *Crypto-Gram Newsletter*, Jul 2001. `http://www.schneier.com/crypto-gram-0107.html`.

[61] Bruce Schneier. Keeping Network Outages Secret. *Crypto-Gram Newsletter*, Oct 2004. `http://www.schneier.com/crypto-gram-0410.html#2`.

[62] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. NFS version 4 Protocol. RFC 3010, Internet Engineering Task Force (IETF), December 2000. `http://www.ietf.org/rfc/rfc3010.txt`.

[63] Robert W. Shirey. Internet Security Glossary. RFC 2828, Internet Engineering Task Force (IETF), May 2000. `http://www.ietf.org/rfc/rfc2828.txt`.

[64] William Allen Simpson. PPP Challenge Handshake Authentication Protocol (CHAP). RFC 1994, Internet Engineering Task Force (IETF), August 1996. `http://www.ietf.org/rfc/rfc1994.txt`.

[65] Theodore John Socolofsky and Claudia Jeanne Kale. A TCP/IP Tutorial. RFC 1180, Internet Engineering Task Force (IETF), January 1991. `http://www.ietf.org/rfc/rfc1180.txt`.

[66] William Stallings. *Network Security Essentials - Applications and Standards*. Prentice Hall, second edition, Nov 2002.

[67] Steven J. Templeton and Karl Levitt. Detecting Spoofed Packets. In *in proceedings DISCEX '03*, 2003.

[68] U.S. Department of Defense (DOD). *Trusted Computer System Evaluation Criteria (TCSEC)*, December 1985. DOD 5200.28-STD (supersedes CSC-STD-001-83), also called the 'Orange Book'.

[69] Sam Williams. *Free as in Freedom - Richard Stallman's Crusade for Free Software*. O'Reilly, March 2002.

[70] Avishai Wool. A quantitative study of firewall configuration errors. *IEEE Computer Security*, 37(6):62–67, Jun 2004.

[71] Robert L. Ziegler. *Linux Firewalls*. New Riders, second edition, November 2001.

[72] Glen Zorn. Microsoft PPP CHAP Extensions, Version 2. RFC 2759, Internet Engineering Task Force (IETF), January 2000. `http://www.ietf.org/rfc/rfc2759.txt`.

[73] Elizabeth D. Zwicky, Simon Cooper, and D. Brent Chapman. *Building Internet Firewalls*. O'Reilly, second edition, June 2000.

[74] Økokrim, Næringslivets Sikkerhetsråd (NSR), and Senter for Informasjonssikkerhet (SIS). Mørketallsundersøkelsen 2003 (The hidden statistic survey 2003). Technical report, Næringslivets Sikkerhetsråd (NSR), 2004. `http://www.nsr-org.no/docs/79281401M.pdf`.