A Personal firewall for Linux

Sten Darre (s030171)

Thesis - IMM, DTU.

30th August 2005

Abstract

This constitutes the investigation and development of a user-friendly tool for firewall-configuration and -management, based on the netfilter/iptables-firewall-code residing in current Linux-kernels.

It aims to be a personal firewall for Linux. It provides a KDE-GUI for netfilter/iptables that lists and edits firewall-rules, lists running processes, and lists connection-attempts to the host. Connection attempts can be accepted/dropped on the fly or permanently handled by manipulation of the firewall-rules.

This thesis constitutes the report-part for obtaining a Masters degree in Computer Science at IMM - DTU.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When this author tried to setup and configure a firewall for SOHO[1]-usage on an SuSE-Linux installation, it took considerable insight into networking and security to see though the tech-hype - and plenty of documentation on issues, configurations and commands had to be digested.

Despite all the efforts, the setup was compromised from the outside in the summer of 2004. The forensics investigation showed that a root-kit (adore-back-door) had been installed and was active. The intrusion was discovered *purely by chance* approximately 2 hours after the breach, when an unexpected network connection was initiated using an inactive user-account (SSH using a guest-account).

Encountering these troublesome efforts, was an eye-opening experience and looking into to this area in general, I found it lacking in usability and difficult to approach for ordinary users.

So, I decided to do something about it - and this thesis: "A Personal Firewall for Linux" describes my attempt:

Designing a firewall-management application for Linux - to be used by *ordinary* computer users.

Much more elaboration on the goal can be read in Sec. 2.4 on page 22, and the specific requirements in Chap. 4 on page 29.

1.1 The Current state of computer security

Having a quick investigation into the current state of affairs of computer security, shows that this is a hot topic these years. Whole books are written about the topic - e.g. [SecCompPP, WilyHacker], and popular press-articles aren't hard to find either. Also readily available, are more scientific investigations and surveys, that tracks and maps the current hacker-threat situation.

However, some of the surveys are showing less transparent results, due to the nature of the them. It is pretty difficult to draw clear cut conclusions from a question like "Where do you feel most at threat from? - the Internet or from within?". By its very nature, such a question is loaded with unknown features and worst fears, like:

- company policies of not disclosing serious breaches (to external surveys)...
- not to reveal or discredit internal policies (as lacking or non-existing) or cast doubts about employees or capabilities...
- ask estimation of the external Internet threat - does anyone really know any hackers out there or their potential...

[1] **S**mall **O**ffice and **H**ome **O**ffice

- reporting 100% of the breaches discovered isn't necessarily the correct answer - the un-discovered and un-impeached don't go on record...

And the list goes on... Indeed, there are surveys that investigates how computer security surveys affect the answers. More on such issues in [WilyHacker, Chap 9.6 - 3rd paragraph].

But, these are the tools available to make an estimate of a) necessity and b) deployment - of computer security. Following are two reasonable surveys conducted from reliable sources. They reinforce some unconfirmed suspected results, drawn from the authors own experience (Sec. 1).

1.1.1 Microsoft survey of computer security breaches.

When Microsoft sent out Service Pack 2 for Windows XP, they made a survey [MS-Survey] to determine the general extend of computer security problems on their platform.

It showed, that **25%** of the Windows-users had been hacked by getting a back-door-program installed, browser-jacked leaving the user with a new home page, phished by being redirected to a fake-website when entering credit-card information for purpose of fraud - or hit by spy-ware that allows monitoring of browsing habits and PC-usage for statistics-gathering in advertising. Almost **40%** had been hit by vira or worms. *All of these breaches where during the last year only*!

According to the same survey, **20%** didn't care about computer-security *at all.* Another **20%** *did* care, but "hadn't got around to do something about it yet". The remaining **60%** did care and had deployed some form of protective measure. The vast majority (>**90%**) of the participants stated the use of their PC "as vital".

This all indicates that the security-measures taken by the users, *does not* match the importance-levels of the users computer-usages. The survey also shows that the security situation for home-users is somewhat lacking - and although surveyed on the Windows-platform - nothing suggests that individual PC's running Linux are differently protected.

1.1.2 Conventional security threat assessment

A survey from Information Week Magazine [SecCompPP, p. 17] show a change in the perceived network tread picture. Commonly, it was thought that 4 out of 5 attacks comes from the inside of an network - ie. behind the traditional firewall at the Internet-hookup.

> Top Methods of Attack
>
> In 2001 , Information Week magazine commissioned a global information survey of 4,500 security professionals. As part of the survey, the respondents were asked to name the primary methods of attack used by intruders against their organisations. (Multiple responses were allowed).
>
> The top method was exploiting known operating system vulnerabilities; almost one-third of the respondents had experienced this kind of attack, The next most popular method was exploiting an unknown application (27 percent). Other commonly used attacks were guessing passwords (22 percent), abusing valid user accounts or permissions (17 percent), and using an internal denial of service (12 percent).
>
> Common wisdom had always been that four out of five attacks on corporate networks or computers were perpetrated by malevolent insiders who could take advantage of their understanding of the system. The survey sought to determine if this "rule of thumb" were true. In fact, with the growing use of Internet applications, outsiders are now considered the greater threat. Hulme [HUL01b] points out that "Many companies suspect hackers and terrorists (46 percent) and even customers (14 percent) of trying to breach their systems." This suspicion is supported by another survey, conducted by the Computer Security Institute and the U.S. Federal Bureau of Investigation [CSI02]. The second survey notes that almost three in four businesses cite the Internet as a point of attack, whereas only one in three cites internal systems.

The noticeable statement is the possible denial of the common rule of thumb: that 4 out of 5 attacks comes from the inside. Threats seems to have shifted some - from the inside to the outside of a network over the last decade. Combined with the proliferation of always-on-connections[2], it puts most users at risk - they are possibly unprotected and available.

The types of methods are traditional though: program-bugs or exploits (**33%**), Trojan-horses or back-doors (**27%**), social engineering or brute-force authentication access (**22%**), and internal threats, misuse or abuse (**17% - 12%**).

1.2 Summary

My personal experience is a reflection of both surveys presented above.

The survey from [SecCompPP, p. 17] (Sec 1.1.2), suggests that the increase and evolution in computer-communications reinforce the need for setting up and maintaining network security policies.

The survey done by Microsoft, as presented in Sec 1.1.1, suggests that it must be easy and accessible too - otherwise it will not be deployed.

The above surveys and my personal experience, shows a need for a management-solution to the firewall-configuration, and that need serves as justification for proceeding the project.

1.3 Project-audience and target-groups

Here, we outline our expected target-group of our work, along with a summary of issues the reader is expected to be familar with.

Report audience The readers of this report are sectioned into three segments, which all must be catered for. The three segments are:

1. Denmark's Technical University (DTU): DTU expects an academic investigation into topics regarding firewalls. The purpose is to demonstrate sufficient skills for achieving the masters-degree from the university. DTU is represented by my supervisor Robin Sharp and the censor(s).

2. Linux Users: Users need for a workable solution. This project is having a pragmatic approach - solutions must bring about something usable for end users. The end users may be any user, with a need for setting up their firewall. The users are *in absentia* and will be represented by the author, supervisor, fellow student etc.

3. The Author: My own need for acquiring programming-skills in areas I fell necessary. The report will bear marks of topics that I found enlightening or troublesome.

<u>Dodgy-clause:</u> The report will not explicitly state where, who's interest is being upheld, but please keep the three parties in mind when contemplating the investigations and solutions in the report.

Product target-group Target groups are the stereotypical user and setup, that is expected to use this software. The typical user in mind are:

- **The novice computer-user**: A user without networking and security skills, but with general working knowledge of using computers - including file-handling (discs), multitasking (processes), Internet usage (connectivity). Such user could be a private home user, any office professional or skilled worker using a computer for business or private use, and now wants to hook it up to the Internet with protection.

  This type of user is generally driven by a simple need: "*just get it working*", and isn't

[2]xDLS connections to ISPs. E.g. ADSL

interested in too many details. A simple overview and as fully automatic algorithms as possible are this group's needs.

- **The experienced power-user**: A professional system administrator, programmer or power user, that have no fear of command-lines and wants insight and control over details. Such a user have an equal desire for easy GUI-interfaces, but usually they have intricate needs and want to control the details under the hood. A extentable and layered software-structure are the needs for this group - allowing tweaking on all levels and in every detail.

1.3.1 Pre-requsite skills and programming issues

To get the most out of the report, insight into the following issues and skills are desirable. Before the task went ahead, some of the knowhow had been previously acquired, and some knowledge needs to be acquired to make the project. First is a list of skills already acquired:

- Linux-OS programming (Bash) and administration skills.
- GUI- and General purpose-programming experience (C/C++).
- Network knowhow.
- Practical software project experience.

And following skills that had to be acquired:

- Firewall realms: Types, setups and netfilter/iptables details (i.e. the actual data to be processed and handled).
- Database knowhow: design (Entity-Relations), implementation (SQL) and technical issues (mysql, postgres, . . .) of databases.
- Kernel hacking and patching.
- Linux API's: I.e. GUI- (Qt/KDE), kernel- (LKM) and special purpose- (libipq, libpgxx,. . .) libraries.
- Specific Unix software tools: autoconf/automake, compiler and debugging tools, KDE-installation and package-management etc. Coming from a Windows-platform the tools aren't that unfamiliar, just different.

The database issue was unforseen at the start of the project, it surfaced during the system-design-phase.

1.4 Readers guide to digestion

The report assumes some knowledge about general computing issues (Processes, Networks etc.) and programming skill (C/C++, APIes etc.). Since many issues are being dealt with in parallel, the report can be read forth-running, interleaved or skipped over some sections, so here's a readers guide to the report.

Firstly, don't miss the end of **Chapter 2** (Sec. 2.3.4 and 2.4) - it contains our mission-statement. **Chapter 4** states our requirements to achieve our mission-statement, and Chapter **6** shows our System Design for implementing and achieving the goals.

Chapter 2 Background into the Internet-structure (OSI-levels), and the definitions of firewalls. Here we establish the foundation of our topic and brings us in position to formulate our project's goal.

Chapter 3 The strategy for achieving our goal. It defines and sketches the software development processes used, including how we discover, define and handle requirements in our project.

Chapter 4 The requirements and features for achieving our project-goal.

Chapter 5 Background into the firewall of current Linux-kernels: netfilter/iptables. The framework, API and the opportunities it leaves. Also currently available OpenSource-solutions and related academic work is investigated.

Chapter 6 The System design of both the end-user-GUI and the core-structure underneath the hood.

Chapter 7 Specific details in constructing the various parts as the project progressed.

Chapter 8 Future work for improving the solution, along with problem-solving suggestions.

Chapter 9 A Conclusion to it all.

It can be beneficial to read **Chapter 2** and **Chapter 5** in sequence, since Chapter 2 is about firewalls in general and Chapter 5 is the specific elaboration of firewalls on Linux.

Also, **Chapters 3**, **4** and **6** can be read in sequence - if good familiarity with Internet-structures, firewalls and Linux is already present.

Chapter 7 gives all the intricate details of implementation, and explains why some requirements are hard to achieve, and **Chapter 8** suggests future opportunities and work on the implementation.

Finally, there is an conclusion to all of this! - in Chapter **9**.

Chapter 2

Background and goal

Let us start by looking into the domain of this project. To the naked eye, the terms may not be a bit fuzzy, so a little history-resume may shed some light on them.

2.1 Firewalls - a computer security issue

The survey (Sec 1.1.1) indicate some user-issues and consequences of not having firewalls installed and configured properly.

Motivation for deploying firewall security Firstly, most users don't really care about technicalities unless they *really* need them. They are not in it for the sake of computers, they are just using computers to accomplish some work. Firewalls are in the realm of networks and computer security, and hence in a *very* technical segment of the domains of computer technology. Users don't want to know about network-devices and -stacks, about packet-switched networks like the Internet (TCP/IP-based) or about good computer security in general.

The only motivating factor is when they have a *need* to know and control it. But the survey shows that, even if the need is recognised, many don't get it addressed. This is possibly to its obscure and daunting technical nature.

Impact of lacking firewall security The impact of e.g. the back-door-problem is noteworthy. It facilitates hackers with new machines to conceal their traffic - allowing spamming, DDoS-attacks and platforms for further attempts of cracking other computers on the net. Therefore, this problem not only affects the individual, but the hole Internet-community as such - by providing platforms for hackers to operate though. A properly configured firewall might not stop a back-door from being installed, but it could disallow communication with the back-door-program - thereby disabling its operation and rendering it unusable.

Firewalls don't check traffic for vira and worms, unless special setups and scanning-software are installed along with the firewall-software - and hence, firewalls don't protect users from vira. But, while vira or worms are not exactly the domain of firewalls - they can make a difference. As with the back-door, they may provide encapsulation of a vira or worm, by e.g. disallowing an infected web-server to send emails or open up any connections[1] - since a server generally don't initiate connections, they only respond. The result is that although the host have been infected, the virus or worm is still being restricted and confined.

The impact of not having firewalls are not only severe for the compromised host, but also *reaching beyond the host* creating threats towards other hosts.

[1] as the CodeRed- and Nimbda-worms did.

Providing firewall solutions As firewalls can look at the network communications, they can both shield against penetrations and encapsulate infected hosts - but only *if they actually are deployed* by the users.

That implies, that users must fell that firewalls are a) *necessary*, b) *comprehendible*, c) *helpful* and d) *effective*. Such is archived when: users have *knowledge* about what firewalls do for them; can figure out how to *operate* them; how to *configure* them right - and finally, that they can *see* the results and *have trust* in them.

2.2 The realms of firewalls

As indicated above, firewalls are helpful to some extend, but to find out exactly *why* and *how*, we must see where it operates. Firewalls are in the computer security domain of controlling traffic on a network, therefor it resides in two areas:

1. *why?* To achieve computer security - presented here by use of the CIA-paradigm.
2. *how?* On the network - presented here through the OSI-model.

The Network-paradigm is more elaborate, since it deals with solutions, and it will therefore be presented in Sec. 2.3.1, but the reasoning through the CIA-paradigm is fairly simple and can be dealt with right now.

2.2.1 The CIA-paradigm

The CIA-acronym is short for *C̲onfidentiality, I̲ntegrity and A̲vailability* as stated in [SecCompPP, Sec 1.3]. Briefly, they cover most aspects of computer security issues - although at an abstracted level.

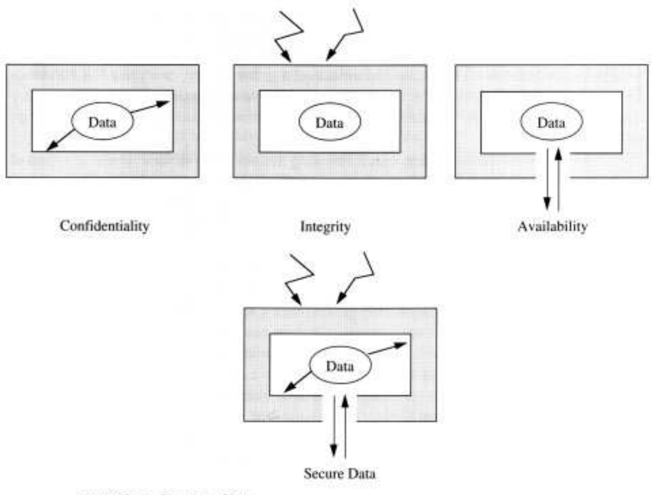


FIGURE 1-5 Security of Data.

Figure 2.1: CIA model.

Confidentiality Ensures assets can only accessed by authorised entities. Sometimes referred to privacy, unauthorised disclosure - aka '*the-reading-of-it*'.

Integrity Ensures assets can only be altered by authorised entities. Also referred to as indivisible, preventing unauthorised modification - aka '*flipping-something-in-it*'

Availability Ensures assets are accessible by authorised entities (when necessary). Also referred to as preventing denial of authorised access - i.e. by its antonym: '*denial-of-service*'.

In all the definitions the term '. . . by authorised entities', which implies: *who* is authorised, *where*, *when,* to *what.* There is no clear cut way of ensuring that someone actually *is* who they claim to be. Securing an identity is a continuing problem.

Cryptography is providing aid in ensuring Confidentiality and Integrity. Confidentiality, by scrambling the data to obscurity, so that only the holders of the de-scrambler are capable of reading it, and Integrity by computing a checksum that is dependant of the unmodified contents and the de-scrambler in uni-some.

Firewalling is not Cryptography - it deals with Availability instead. That is, allowing communication passage through the wall. However, firewalls are not *ensuring* identification, just it uses identities of the communication to make decisions about the passage - hence, if the communication is identified as authorised, the traffic will pass through. And getting the identity right is *a major* issue for firewalls.

2.3 Firewall- and Networking-terminology

Before we can figure out how firewalls actually achieve these results, we must take a look at how they operate. That is topic of the following sections. But first some very basic terms and definitions..

Firewalls - how do they achieve results How does a firewall operate to achieve the above desired results?

A firewall looks at the network and does it's magic there. It is a well know technology, which have been around for many years. It operates on particular sections of the network and to identify these sections, we first discuss the overall context of networks. Many of the following illustrations have been shamefully ripped from [ApplComms, Chap. 2 & 5]

2.3.1 The OSI-Model

The academic foundation of network communications is expressed in the OSI-7 model. It is depictured in Fig 2.2, and contains 7 layers numbered from the bottom, i.e. Layer 1 is the Physical layer and layer 7 is the Application layer.

When a program-application wants to send some data, it starts off at the top of the layered stack (to the left). Each stack models the network paradigm on one host. Each Layer in the model then receives data from a layer above it, add it's own pre-pended administrative header and pass it on to the next layer down the stack. In the receiving end the reverse is done. This creates the illusion that each layer is communicating with the corresponding layer on the stack of the other host, as indicated by the arrows across. Since each layer strips off it's own data before parsing it upwards (and downwards they haven't even been added yet), a layer above cannot see any layers below, they appear transparent - hence the virtual data-flow.

The scope of these layers extends to the physical worlds as illustrated by Fig 2.3, where the closer to the bottom layer, the closer the model maps to actual bits on a particular wire and the less the abstraction is. In the figure, an example is shown of two nets with of four computers each. Each net resides on its own network-type (e.g. an Ethernet-net and a Tokenring-net) i.e. N1-4 and N5-8, and the two nets are connected through node N2 and N5, which may be connected by e.g a modem-line.

We will use this type of setup shortly to display the Internet in OSI-terms since it is a very stereotypical and commonly used setup.

We will refer to layers of the OSI-model through out the report, they are our foundation and is implicitly used throughout the domain of networking and firewalls. However, in our context, the OSI-model itself is only interesting with respect to the Internet. Next, we'll look into the layers functionality and their placing with respect to the Internet.

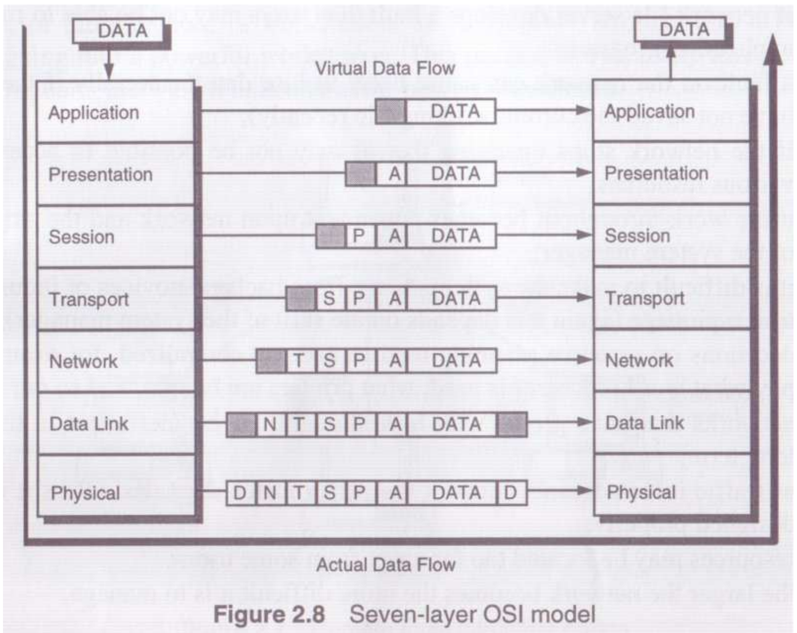


Figure 2.8 Seven-layer OSI model

Figure 2.2: OSI-7 model.

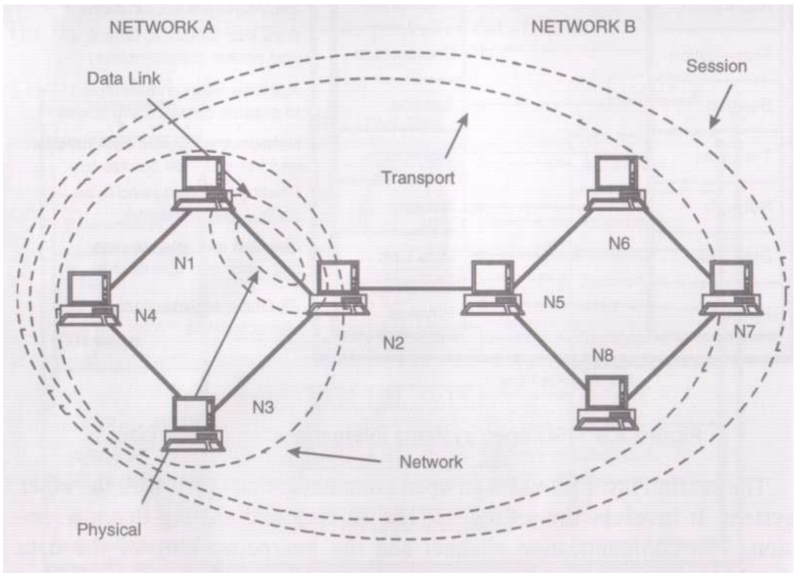


Figure 2.10 Scope of concern of OSI layers

Figure 2.3: OSI-7 model physical extend.

2.3.2 The Internet in OSI-terms

The Internet architecture was conceived before the OSI-model, but the OSI-model is capable of mapping most protocols to it's layers. Examples of various network protocols are shown in Fig 2.4, and they include the Internet (TCP/IP), Novel's Netware (IPX/SPX) along with some familiar hardware devices like network-cards (Ethernet and Tokenring) and serial-lines (modems).

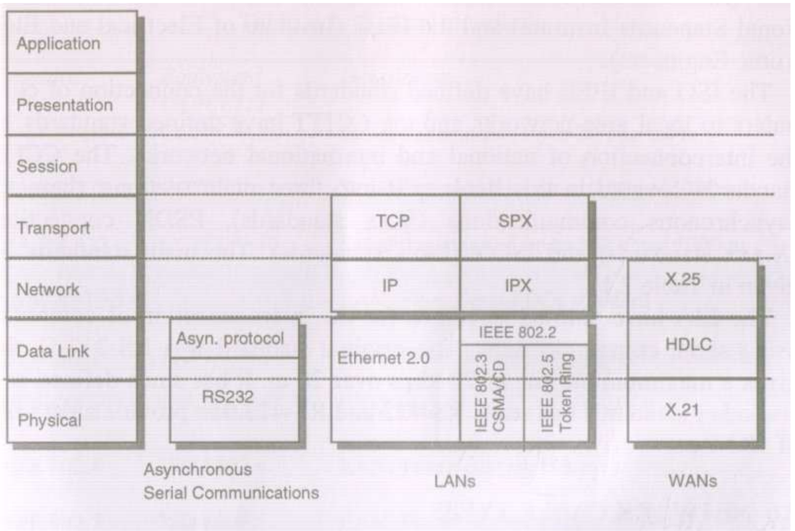


Figure 2.11 ISO open systems interconnection (OSI) model

Figure 2.4: OSI-7 vs. Internet.

To the applications the Internet looks like a transparent virtual wire between the connected end-points. But that isn't actually the case, as we shall see next.

2.3.2.1 The wiring of the Internet (Layer 1-2, Physical-DataLink)

The task of physically making error-free transmissions of bits from host to host, is done by communications-hardware and their OS-device-drivers. The type of device can be wired (electrical copper-wires or optical fibres) or wire-less (radio-communications), and the physical- and datalink-layers will change and adapt accordingly. Each device can only communicate to other devices of its native physical type, like satellite-to-satellite, ethernet-to-ethernet or modem-to-modem etc. This is illustrated by the two (physically different) networks in Fig 2.3.

In order for the Internet to span all types of networks, it must handle how to span several different segments of hosts - because the physical devices don't. If the Internet didn't handle it, data couldn't cross over from e.g. a modem- over to a fibre-connection - as illustrated in Fig 2.5. Connecting dissimilar network-devices is the task of the next layer: the network-layer - and for the Internet, named appropriately: InternetProtocol-layer (IP).

2.3.2.2 The Internet *does* comes in Packets (Layer 3, IP)

The Internet it self, is a *packet*-based network[2] which is a best-try, connection-less network, where - in theory - all parties share the same transmission media, i.e. the "wire" (Layers 1/2). The net is equal to all participants and they rely on each other to relay the channels back and forth. As

[2]An opposite example of the Internet's packet-based network, is the telephone-system - aka a Public-Switched-Transmission-Network (PSTN) which is a guaranteed and connection-oriented network. There, a dedicated wire is established between the two end-points, and also, it is always clear when a connection is established (<ring-ring>), used (talking) and torn down (<click>) - guaranteeing the channel's connection-state (and bandwidth).

hosts and routes may be up or down at any given time, a transmission is hoped to make it through as gateways along the way give it their best try.

Since all hosts share the same media, some form of addressing packets are required, and the solution is a model resembling the real-world postal system of delivering ordinary mail and packages. Each packet is wrapped with a header at each layer providing the addressing needs of that layer - in theory. In practice, some implementations don't need addressing schemes - e.g. a modem-connections on layers 2, only have two endpoints, so no need for addressing for level 2.

Elaboration (technical) as to why the design is packet-based: If multiple communication-channels are to co-exist simultaneously and by-directionally (duplex), all while sharing the same wire, then two options are available: 1) time-share the wire (Time Division Multiplexing) or 2) frequence-share the wire (Frequency Division Multiplexing). Both can be deployed at the same time, and probably is through out the distribution network.

Technically, it implies that the communications must be parted into "*slots*" of communication before sent out on the wire. An example of time-sharing the wire, is Walkie-talkies - only one participant obtains the channel at a time, i.e. the participant is assigned a time-slot. An example of frequence-sharing the wire, is the FM-radio-band - multiple participant, each on their own channel, i.e. assignment to a slot in the frequency-spectrum (a channel in electronics lingo).

The Internet is capable of both, with multiple channels of single participants on each channel. The data is sectioned into *packets*, leaving it to the electronics-domain to decide what type of multiplexing (time and/or frequency), since packets can easily conform to either type of slots.

2.3.2.3 The virtual wires of the Internet (Layer 4, TCP/UDP)

More often than not, the desired type of communications is the way the telephone-network works - with its connection-oriented features. Therefore, an extra layer is necessary to keep track of the connection state and intermediate communication flow. It's task is to establish a *channel* between any programs that wants to communicate together. It does so by providing a *socket* for the program to connect to. It chops up the data-streams from the program into packets before tranmission, and reversely - to reassemble received packets into data-streams, before handing the data over to the program.

2.3.2.4 Internet programs (Layers 5-7, any protocol)

The applications only see finished (re-)assembled data-streams to/from a socket. They request the socket to be connected to some program on an another host, and once established, they transmit and receive data from the socket - as it was a serial wire between the to programs.

2.3.2.5 An example of Internet communication

The OSI-model's usage is illustrated in the following example (see Fig 2.5): Two programs on two different hosts connected through the Internet, wants to communicate with each other using some channel - in the example, it is noted as an FTP-session, but the particular type of session isn't important here. The two host is connected through intermediate hosts along the way - named *gateways* in the illustration. Their purpose is to forward packets in the right direction. Traditionally, a gateway would also perform the role of a firewall in a company - as described earlier in Sec 2.3.4.

The channel is established by a computer on network A towards a computer on network B. Once up and running, computer A can send packets to it's Internet-gateway addressed for computer B. The gateway makes some routing decisions as to which wire to pass on the packets and winds up sending the packets in the right direction - as illustrated by the middle OSI-stack (Internet gateway) in Fig 2.5. The bottom figure show the path of packets as they traverse the various OSI-layers on computer A, the gateway and computer B respectively.

The packets are handled by each layer by adding that layer's handling-data (called *a packet-header*) to the packets when they pass through. Fig 2.2 shows the amendments of bits as the

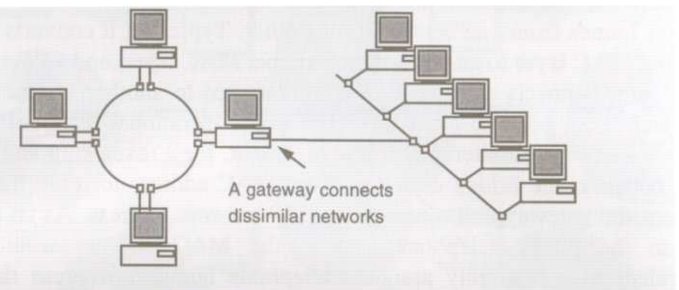


Figure 5.2 Routers and gateways

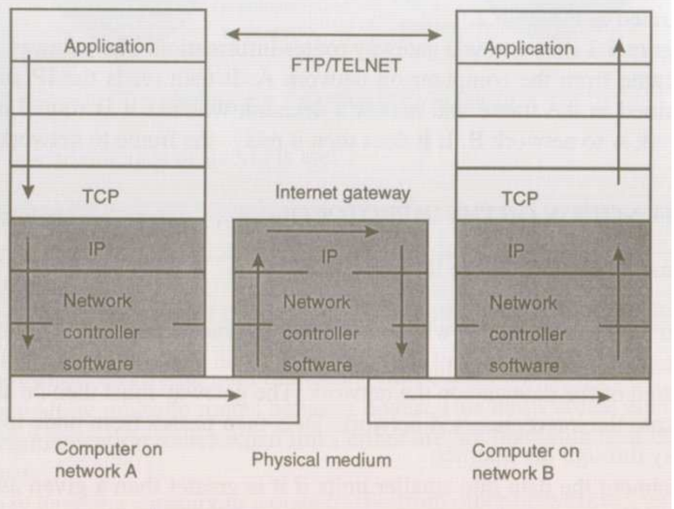


Figure 5.3 Internet gateway layers

Figure 2.5: OSI-7 example session.

packets are put through the OSI-model. When packet traverse upwards again, each layer strips of it's header (handling-data) before parsing the packets on to the level above.

2.3.2.6 Summery of the OSI-model of the Internet

In practice, the layers 7-5 is application-specific and they may operate as they see fit, e.g. mail-applications do SMTP-wrappings, browser do HTTP-wrappings and "Joe's Download Applet"s may what ever wrappings it sees fit - it's just data. These layers contain the actual data-contents.

Layer 4 is taking care of the routing to/from programs on the host and the chopping and re-assembly of the programs data-streams. So, they add the TCP-header or equivalent header for this type of transmission protocol. e.g. real-time-protocols (RTP) could add some expire time-stamp and bandwidth requirement etc. This layer have the information of what program is involved in the communication and what state the communication is in (e.g. new <ring-ring>, established <talking…>, or down <click>). In particular it wraps source- and destination-ports, where a port is one endpoint of the channel/socket.

Layer 3 is taking care of the routing to/from hosts and chopping/reassembly of packets to fit the bandwidth-size of the physical layers underneath (e.g. modem, Ethernet, satellite…) - packets in this context are referred to as a *frame*. This layer have the information about which computer/host is involved in the communication. In particular it wraps source- and destination-IPs.

Layer 2 is the actual hardware-device performing the communication and its primary task is integrity of electrical transmission. That is, to transmit and receive bit on the transmission media (wire, antenna, fibre…), while ensuring the transmission is free of bit-errors due to noise, interference and other naturally or human induced physical problems. It isn't concerned with what it transmit - only with it's error-free delivery. As such, this layer doesn't contain much information concerning our firewall-task, but it wraps CRC-checksums[3], MAC-address[4] and other stuff around the packets from the above layers.

2.3.2.7 Key points

OSI-7 ***O****pen* ***S****ystems* ***I****nterconnection*-model. The theoretical model used in network communications-theory.

Layer 7-5 Application- Presentation- and Session-layers. The program-applications part in the OSI-model. Very specific to the type of program.

Layer 4 Transport layer. The (packet) transmission control part in the OSI-model. Responsible for handling packets to/from programs.

Layer 3 Network layer. The routing and (frame) transmission control part in the OSI-model. Takes care of the host-part of routing and handling the packets.

Layer 2-1 Data Link- and Physical-layers. The electrical transmission part in the OSI-model. Responsible for transmitting frames without bit-errors.

channel The communication-media used for transmission of data-streams between programs - i.e. "the wire" between the two programs

socket The program-endpoints of a channel established between two programs on two hosts - i.e. "the plug of the wire"

packet A lump of data (with headers) transmitted through a channel. On lower levels referred to as a frame.

header The addressing part of a packet - i.e. "the envelope" of "the package" (stamps not included…)

[3]Cyclic-Redundancy-Check

[4]unique id

source- and destination-address The parts of the header, which specifies the two endpoints of a channel.

ip-address One end of a channel, specifically identifying the participating host - usually represented as a number. Actually, it identifies the physical hardware-device inside the computer, e.g. Ethernet-card number 2, or serial port number 3 etc.

port-address One end of a channel, specifically identifying the participating program on that host - usually represented as a number.

2.3.3 Firewall-types

Different types of firewalls exits, depending on which layers they operate on and how thoroughly they check. Mostly, the following types seem to exist, but the definitions are a bit fuzzy and overlapping depending on which source or book is referenced. Below are the definitions - mostly inspired from [LinuxFWip, Chap 2]:

packet filters These are firewalls that have rules that covers what connections are allowed to pass in or out - based on ip- and port-addresses. These operate on layer 3 and 4, and look at one packet at a time. They assume no affiliation between packets, and hence, they are *Stateless.*

application gateways These firewalls operate on the upper layers too - making inspections of not only the packet-headers (layer 3-4), but also the packet contents (layer 5-7) - if they can recognise the contents. They must have specific application knowledge in order to check the contents.

proxies These firewalls take the application gateways all the way, by not just monitoring the packet-contents, but replacing themselves as the connection endpoints.

circuit-level proxy These firewalls are proxy-ing by replacing themselves as the connection endpoints, but they *don't* inspect the packet contents. Instead they may ask for authentication before allowing connections to proceed.

dynamic packet filters Also known as *Statefull* firewalls. They can be seen as mostly identical to ordinary packet filters, but they keep track of the whole connection (the channel) - and not just one packet at a time, thereby, they can expect and predict incoming responses to packets just sent out.

<u>An analogy might be presented here</u>: If you (a client) want to visit an inmate (a server) in a prison (protected network of hosts), you will be subjected to restricted behaviour by the jail-authorities (firewalls) in order to uphold a security policy of not discussing or bringing escape-methods (no files baked into cakes etc.) to the prisoner.

The firewalls may work in the following ways:

- **packet filtering**: You show your identity papers at the gate and then - if allowed - go see the prisoner in a un-supervised room.
- **circuit-level proxy**: You show your identity papers at the gate and then - before seen the inmate, you are to be cleared again by an officer who personally knows you and your relation to the inmate - before the visit is allowed in a un-supervised room.
- **application gateway**: The room is now supervised and the conversation is tapped, e.g. the scenario of separation by glass with two telephones to talk through. What ever you say may be discovered, but goes unimpeded to the inmate anyway - before you are thrown out...
- **proxy**: Now you don't see the prisoner directly, the conversation is terminated through a prison officer. Now you talk to the officer which then relays the message to the prisoner. What ever you say may be halted - and now, the message does not reach the inmate.

- **dynamic packet filters**: Now they keep track of your visiting habits, how often, when, how long etc. Any deviation from the familiar pattern is halted. But they still don't know what you do - only the pattern by which you do it.

All of this is presented more in detail in Sec. 5.1, but for now the current Linux-kernels (V2.4+) can do just about all of them - that is, if they are configured right.

The existence of so many different types of firewalls sterns back to the elaboration of the CIA-principle (Sec. 2.2.1). Firewalls are dealing with *availability*, and in that statement it is implied *to whom* - meaning *authentication*. But the Internet is essentially an flat, even, anarchistic, best-try network (Sec. 2.3.2.2) and it wasn't designed for guaranteeing authentication *on* the net - only availability *of* the net. Therefore, there are many ip-addresses and -ports with commonly presumed or expected programs on them. But in reality, <u>they</u> <u>are</u> <u>unknown</u> - <u>until</u> <u>they are explicitly identified</u>, <u>authenticated</u>, <u>verified</u> etc.

In essence, all of these firewall-types aspire to live up to an old russian saying (political - from the Kremlin):

Trust, but verify... [WilyHacker, p. 103],

which loosely translates into:

Trust is good - but control is better...

In order to achieve *'... better control... '*, all of these above firewall-types needs to be deployed on each host - and easily too. The subsequent definition of "the personal firewall" will form the bases of our project-mission.

2.3.4 Definition of "the personal firewall"

A "Personal Firewall for Linux"... well - what is a *firewall*?, what is so *personal* about it? and more importantly: why would somebody need one? To answer these questions, we start off with the need.

The need arise when a computer is put on a network, and thereby is given the ability to communicate with other computers. Then, we need to setup an *access-control* on the communication. Otherwise every communication-type from any computer is allowed - leaving the computer potentially wide open. And this is usually *not* what the user intended. Or worse, the user might not even be aware about such issues at all, and could remain unaware about the openness, because it cannot easily be see on-screen, that the computer is communicating though some wire.

<u>Deduction</u>: The mere visibility to the user (of network-control) is an issue by itself - *seeing is believing...*[5]

The solution is to deploy a firewall, that can enforce some access-control on the communication. It only a first line of defence mechanism, not a full access-control system. It generally does not authenticate users on behalf of the system (like e.g. a circuit level-proxy), it only checks hosts and programs, i.e. its a packet-filter (statefull or not). And that is done a by scheme, which isn't too bullet-proof - as 'indicated' by [LinuxFWip, Chap 2], [WilyHacker, - most of the book!] and [SecCompPP, Chap 7].

[5]Last experienced, when de-entangled the rules in a 'ZoneAlarm'-firewall on a Windows-box for a friend. As the firewall was switched back on with logging of denied attempts, he cried out every twenty-second or so: '*...Wow - there's <u>another one</u> trying to get in...*'. He was stunned at first - and left elevated by the significance of the firewall. His computer-security worries started right away: '*..Jesus - I've got home-banking - can they have gotten to my bank-account ??..*'

The firewall is given a proper configuration, which is specifying how access-control is to be enforced. This is generally referred to as a *security-policy*, that specifies what type of communications and which computers are allowed to communicate with each other - i.e. the rules of the firewall.

As such, it resembles the need of e.g. usernames and passwords - its just a computer security issue, which most users only have a marginal interest in... that is, until the security have been compromised and harmful, undesirable communication have occurred.

In brief, handling firewalls (and their rules) are fairly unwanted and ill-favoured by most users - but firewalls establish the first line of access-control to a host and the network it resides on.

So, a firewall is a solution to our need for access-control on the network. But why a *personal* firewall? Although users tend to take security breakdowns *very* personal, that isn't where the naming sterns from. The term refers to a more recently evolution in the Microsoft Windows-world, where client-users are referring to their firewall-software as *personal firewalls*, implying a firewall *for this host only.*

Traditionally, no firewalls where deployed on home users or client machines on a local network. The network was considered a trusted environment and the only protection was the *access-point* (router/gateway) between the local network and the Internet. This access-point was enforcing the access-control on the communications - aka enforcing the security-policy. In such scenarios, the access-point was referred to as *the firewall* protecting all machines within the perimeter against harm (*... hmm - fire?*).

No user had any actual interaction with such a firewall - they couldn't see it - and most users never even knew it existed.

But along with the personal windows-firewalls comes the user-friendliness of the Microsoft platform: Suddenly, the firewall can be installed, checked, altered and seen processing the communication of network packets and thereby becoming very visible to the user. Occasionally, it might even pop-up dialogs on the user's screen asking for confirmation for allowing a particular communication-session by a particular program.

Also, a further mixing of terms from firewall-types becomes customary: Most of them are just packet-filters, but they know about which program are involved in the communications - although they have no actual application-gateway- or proxy-capability. They do not inspect the packet-contents, instead they trace the end-station of the packets to the actual program. Hence - they *appear* to be more intelligent than mere packet-level filtering - but in reality, only the authentication of the host-/program-pair have changed: from ip-/port-addresses-mapping to socket-/process-mapping.

The cross-bread can be classified as a ***circuit level-packet-filter***, since it mostly does what a circuit-level proxy would do, and it operates on the packet-filter-level (OSI 3/4). But differs from a proxy on several issues: Firstly it does not proxy the connection, it only stalls the packets on the network-stack (awaiting a user-decision). Secondly, it does not ask for authentication at the client end of the connection - it asks the currently logged-in user on the firewall-host instead. And the authentication is not e.g. a key or password, it is a visual (dis-)approval of an running program on the host.

As such, the term "*personal*" might have started off as meaning "*for this host only*", but currently it also imply high visibility to the user along with user-interaction concerning the access-control. Hence, the term "*personal firewall*" implies the whole setup of security-policy along with any interactions regarding this host and the network it participates in.

Relaxing the meaning of "*for this host <u>only</u>*" make even more sense from a Linux point of view, due to the general nature of Unix-type operating systems being very network-centric. Today, very few computers operate alone and most hosts gets exposed to networks.

2.4 Purpose and goal

Having Microsoft's computer-security survey form Section 1.1.1 in mind, the development of an automated, foolproof algorithm for the settings of a firewall, would protect *all* 100% of the user-base from harmful computer-communications - including the 20% that didn't care at all.

We will have an more pragmatic approach in mind, since such algorithms haven't materialised easily yet: A tool that sets up good firewall-filtering using fully- or semi-automatic algorithms, thereby aiding the user as much as possible. That wouldn't cover the 20% which simply don't care, but the 20% whom haven't '*..gotten around to it yet..*' might feel animated to *get going*. Finally, the last 60% of users, gets a much improved customisation and flexibility with the advert of better tools

The goal of this project is to:

1. Establish the framework for firewall-configuration:
   That necessitates an application-framework that allows better and easier handling and manipulation of iptables-firewalls. Such framework would include an establishment of an iptables-API and a GUI-framework for algorithm-modules to exists in - and thereby also supplement each other.
   The hope is to leverage module-development and let modules gain from one another by simpler and easier deployment - i.e. a structure to place an algorithm or user-functionality into. That implies a decentralised build- and release-structure, where modules can be build independently of the application-framework.

2. Investigate semi- and fully-automatic solutions for user-control of the network-access:
   The ***circuit level-packet-filter***-hybrid-type as known from the Windows-world, is a reachable semi-automatic network-access method controlled by the user.
   Some wizard for setting up initial firewall-rules, taking into account the network-interfaces and network-layout - in addition to standard services to open up (or close) for.
   Any semi- or fully-automatic solutions starts off with roundtrip-engineering as a foundation. That is: Importing the rules, modifying and improving them - and then exporting the back into the firewall-host. Otherwise users cannot get the benefits of experienced setups with continuing customisations as time passes by.

All development is done with a focus on pragmatic usability for the average end-user (hence the friendly 'buddy-buddy'-term: *personal firewall*).

Chapter 3

Project prelude - Engineering strategy

In software projects some tasks are re-occurring, what ever the domain is. Before progressing with the project itself, some consideration as to the working-frame and -methods are beneficial.

In this Chapter, we outline our work-process used throughout the project, finishing off with a summary of our requirement-template.

3.1 Processes and procedures

We will deal with the forthcoming project by adhering to a well-tested and -tried recipe - a development-process model. If the recipe doesn't suit the project well, trouble is likely to occur. The solution is to alter the recipe and not the project, so our chosen model will be a cook-up of known models - a composite model. Many recipes exists and below we outline the core ingredients of our selected model.

3.1.1 Development-process models

We will draw upon these abstracted and idealised process-models (taken from [SW-Eng.(7ed), Chap 4.1]):

1. *The waterfall model* This takes the fundamental process activities of specification, development, validation and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing and so on.
   Properties: Structured, formal (though not to an mathematical extend), bureaucratic/rigid, usually embodies a hierarchy, traceable.
   Advantages: Well-known and well-tried, been around since the 70'ties and is the traditional approach. Produces thorough amount of documentation.
   Disadvantages: Linear model with forward-only progressing activities. Inapt to changes during the project. Very detailed planning produces inflated amount of documentation (since back-tracking is problematic). Slow (in comparison).

2. *Evolutionary development* This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from abstract specifications. This is then refined with customer input to produce a system that satisfies the customer's needs. (*Evolutionary development* is also known as the *Agile methods*: XP, DSDM etc.)
   Properties: Darwinistic approach and property of code, naturally inclined to cowboy-programming, very flat hierarchy.
   Advantages: Inept to change by producing code as needed through natural progress. A Darwinistic property is inherited in the software - nothing gets developed or optimised without

a need, and old unused code dies. Fast. Scales well.
Disadvantages: Difficult to maintain status and hence overview and track of progress. Large-scale projects may grow wild. Much less documentation of progress, state and possibly of the resulting code-base.

3. *Component-based software engineering* This approach is based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them from scratch.
Properties: Shopping based, compound structure, experienced based.
Advantages: Reuse, fast, less own development and re-inventions, less risky.
Disadvantages: May not perfectly fit the tasks with the selected components. Have a tendency to develop ”add-on”s, bulges and other artifact growth.

All of them have their own problems and advantages. But, as always, we'll avoid the problems - now that we are aware of them. This project can draw on all three models due to the following observations:

1. *The waterfall model*: It is used everywhere, most businesses organisations adhere to this model to some extend. Even the report contents-layout may have traces of this model: background-requirements-design-code-test-conclusion or in a business cycle: analysis-bid-contract-specs-implement-deliver-payment.
When: This is well suited when the task is well defined and clear - as it should be when implementing a particular module or sub-part of a project.
Why: tradition (!?), because at the code-level, development-tools like compilers, debuggers, scripts, etc. are created with the waterfall model's tasks in mind ?
Where: This is the process to use during actual development: specify a module - code it - test it - document it.

2. *Evolutionary development*: It is used by e.g. the Open Source community - including the Linux-kernel itself. It carries a Darwinistic approach and property with it, since nothing gets developed without a need being discovered first. Old code becomes obsolete and incompatible when unused, due to evolution of surrounding code, or it gets fixed and upgraded because it is used and found needing a patch.
When: This is good for discovering a topic, for prototyping and for incremental growth in stages of a system. When speed is desired.
Why: We are already building on an existing code-base which is the incarnation of this approach. Additionally, it isn't clear exactly how a good user-interface should be. Likewise, building upon the existing kernel and GUI-libraries may dictate some circumventing and changes.
Where: During the the unclear stages of e.g. usability, man-machine interfacing, kernel-modules etc.

3. *Component-based software engineering*: Used by businesses in sectors where ”the shovel” have already been invented. Like, databases-implementors, web-companies, advertising, entertainment industries etc. They don't make databases, web-servers or rendering programs themselves - they use the programs, and add the needed glue-modules to customise them to the job at hand.
When: To leverage development by standing on the shoulders of other developers code. It carries an inherit reduction of risk and complexity since less own development for the same amount of functionality is necessary.
Why: We are already building on an existing code-base: the firewall in the Linux-kernel (netfilter) is component based by it design, the Qt-GUI library (KDE) or other GUI that we'll use.
Where: All the things we can steal, borrow, lend or otherwise reuse by applying more or less glue. Only the modules we want to invent are outside this model.

In brief: We'll look at it component-based, do evolutionary development of overall design and modules, and in each cycle commit to the waterfall-model.

3.1.2 Process iteration

Most projects, including this one, will be subject to constant discovery and changes as the project makes progress - as stated in [SW-Eng.(7ed), Chap 4.2]. Therefore, an evolutionary approach is required. In our home-brewed compound model, the individual stages of the three processes are traversed in a iterative pattern as necessary. If the traditional waterfall method is successful, an linear incremental pattern is present: development and delivery of base-modules and then completing one module after another according to dependency. However this is rarely the case, due to unforseen circumstances that requires re-thinking and back-tracking in the project.

To avoid such, the spiral traversal of the development-model is used - as illustrated in Fig. 3.1. It suggests the four quadrants of (start lower left - CW): A) define and refine planing, B) specifications of needs and hows, C) analysis of risks and prototyping and finally D) implementing the stuff. The implementation-part are the actually the traditional linear-forward waterfall-model of: a) requirements b) design c) implantation and d) verification.

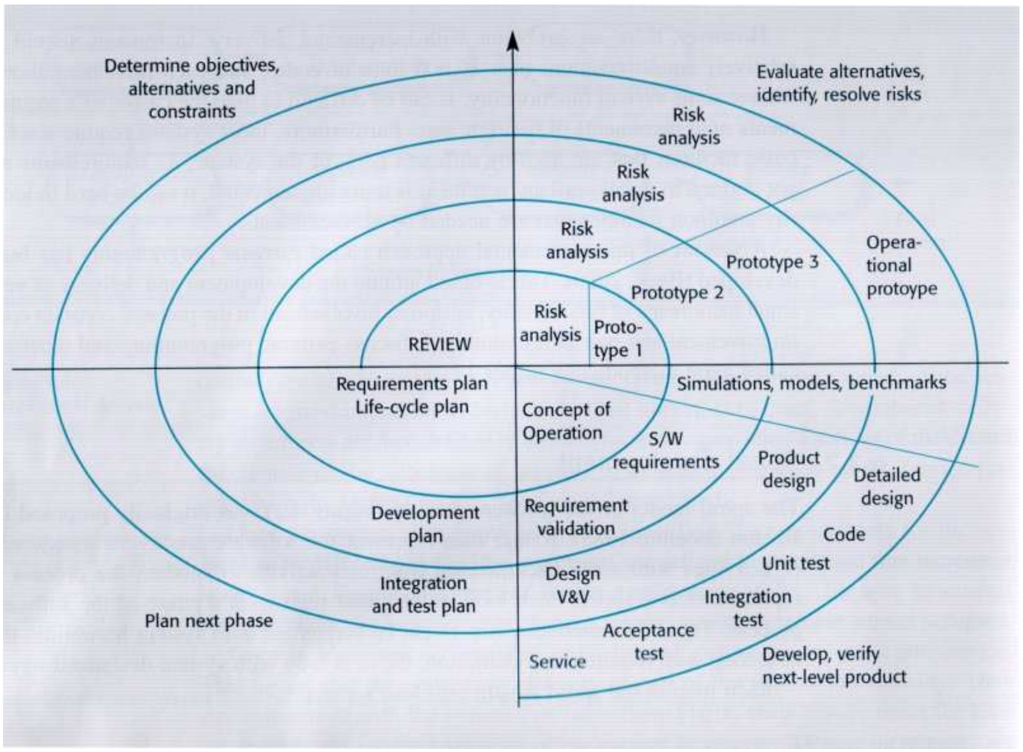


Figure 3.1: Iterative development-spiral.

The spiral contains elements from all of the three models above. One could do an overall waterfall-model by having only one swing in the cycle, and do the implementation-part by using an agile method like XP. But we don't: Overall evolutionary model with waterfall-methods in the details - resulting in XP/DSDM-discovery and planning of new modules, glue upon existing components with several development-phases, each as a traditional waterfall-section.

3.2 Requirements handling

In order to get the project started on the right track, the traditional waterfall-model gives us a rough and well-known road-map:

Conception -> Requirements -> Design -> Implementation -> Verification -> Done.

The conceptional phase setting us up for this project was the personal experience introduced in previously in Sec 1. Next comes the requirements, design etc. which all suggests the traditional waterfall development model, but it need not be so traditional.

By working with the requirements, a more flexible and finely tuned composite-model emerges. Different requirements can be sorted and grouped, and through that process, we can identify and use either a more suitable overall development-model altogether (E.g. Waterfall, XP. . .) - or just some detail-methods for specific development in sub-areas (E.g. ER-database-design, Automata-parser-design. . .).

One can sort requirements accordingly to their origin and level of detail. Some sorting into groups are suggested in [SW-Eng.(7ed), Chap 6] and these are:

1. User requirements: Overall descriptions of end-goal.
   Target group for this information is: System-architects and -end users, Contract-officers and -managers and others who are managing the overall specifications of the system - at an more abstract level.
   (a) Natural language descriptions of what the system is to provide - i.e. the idea/goal - in order to bring about the full picture. It is possibly supported by defining constrains, illustrating with sketches etc.
2. System requirements: Detailed specifications of need
   Target group for this information is: System-architects and -end users, engineers, developers and others who are involved in the construction of the system - at a more detailed level.
   (a) *Functional* requirements.
   Specify *what* services and functionality the system should provide. Rarely, they may state explicitly *what* functionality that *will not* be provided, in order to avoid hidden expectations to some implied functionality.
   (b) *Non-functional* requirements.
   Declares *constrains* on services and functionality that the system will provide. Limits the *extend* of the services that will be provided. They can be timing-, platform- or functional-constrains.
   (c) *Domain* requirements.
   Inherited and implied requirements from the domain of the system - the application area. The *'usual-things-one-can-be-expected-to-do'* with this type of system.

As with any intuitive natural grouping, opinions may vary as to which group a requirement naturally belongs to. But its individual placing isn't the important topic here - it is the process itself. It creates an process for:

- <u>Establishing an overview</u> by handling, categorising and organising the requirements.
  This way, identical requirements get collapsed into one and sometimes each group of requirements also end up being implemented by the same sub-module in the final the system.
  At the grouping-process, requirements also get split up into several when they are overlapping several issues and new issues gets discovered when dealing with the requirements.

- Finding suitable methods of managing the development process in later stages.
  I.e. by allocating a process to those groups for which an established well-structured development-process already exists - like ER-database-design methods for any database-grouped requirements.
  Other groups may be at bit more vague in structure - like usability - and simply identifying them as *out-of-ordinary-category* helps by not enforcing an unsuited development process over the issue.

The process itself *cannot* be seen in the report - only the resulting categories. And the categories aren't very spectacular, it's the process behind it, that does bear fruit, by organising the tasks for the implementors.

3.2.1 Requirements template

Req#	*Description*	*Reasoning*	*Category*	*Priority*	*Workload*
...	...		...		
27	The User can click on...	Because of visibility, ease with...	Functional GUI	Essential	Medium
...	...		...		

Table 3.2: Example of Requirements.

In table 3.2 an example of a requirement-description is shown. It contains the result of a requirements-handling process, and contains the output of various stages from the development-spiral.

The first column contains a requirement-number, which is just a serial-number, but it is added to ease referencing and enable some tracking of requirements - e.g. "*...requirement #27 is an elaboration of #4 with added traces from #17, which makes #4 obsolete and it will not be implemented...*" etc.

Next comes the description and the reasoning, i.e. "*the what and why*". This is where the User- and System-requirements are specified.

Last column contains the assessments from the analysis- and risk-phases of the development-spiral. Together they form the bases for a plan of implementing these requirements. The column may additionally contain information of dependencies to necessary existing modules - but that isn't necessary in our case, since our most of our dependencies already are existing.

Apart from the categorisation-process discussed before, requirements are also given a workload-estimate [Easy, Medium, Hard, Unknown], and an importance-weight of either [Essential, Highly desirable, Desirable]. The workload-estimates are very rough initial estimates, but they do give an idea about the task ahead.

As for the importance-weightings - if at all possible, the [Essential]'s should be implemented. The [Highly desirable]'s are just that - non-essential - the solution will still be functional, but look pretty bare-boned without them. The [Desirable]'s are nice features and will implemented if time and effort permits.

3.3 API-Coding strategy

The actual implemnetation and coding requires a bit of attention - strategy-wise. The project is likely (hopefully) to contain some modules developed by 3. parties. This requires an interface for plugging in such a module - more on the exact details later on.

The point in this section, is that we need to find out whether or not a good solution for plugins have been made. To do so, we will design and use the plugin-system for all our own parts too, thereby testing the interface and associated procedures for making 3. party extensions to our application.

In essence: We will be trying our own medicine on ourselves.

The consequences are, that the interfaces will have to be established early in the project, since our own development will suffer from a 'moving' interface too, so it has to be stable early on. The benefits are a better chance of getting the interfaces right.

Chapter 4

Requirements

Being animated by my personal setup-experience (Section 1) and finding justification in the surveys (Section 1.1.1 & 1.1.2), it is time to outline some success-criteria for resolving the problem. Firstly, let's define the **problem** - then establish a list of success-criteria for the **solution**.

Problem The problem revolves around the *configuration* of the firewall - mostly. The actual firewall framework established within the Linux kernel is quite competent, allowing very capable and versatile firewall-solutions to be produced using this framework.

But it is the process of how to setup and maintain the firewall configuration which is problematic. The process is obscured by complex topics of network, routing and security that inhibits ordinary users from creating good setups without aid. No current solutions was found helpful enough for ordinary users - they all requires modest programming skills and a good network knowledge to operate the solution correctly.

Solution Our solution must change this, allowing users with no programming skills and little network knowledge to operate the firewall. Meanwhile, our solution must not discourage users with programming skills and good network knowledge from using our tool, they should have an equal lift in opportunities.

The work diversifies into several directions:

1. Present a solution of a GUI-part that allows the user to control the firewall with ease (improved usability-UI over the existing solutions). Very Usability-oriented.
2. Construct the necessary missing modules, enable-ing the GUI to interact with the firewall at runtime (none exists currently). Possibly, some new sub-modules in the kernel are needed, in order to make the solution interactive and add the "windows-like personal-firewall"-solution along with the *circuit-level-packet-filtering* (see Sec. 2.3.4).
3. Create the necessarily layered API allowing expendability, modularity and versatility, housing the framework for GUI-modules that presents (*overviews*), aids (*wizards, dialogs and settings*) and checks (*verifiers*) firewall-configurations. This API carries some implicit demands of a more technical nature, relating to the build- and release-dependencies.

4.1 The list of requirements

Requirements found for this project are categorised (as described in Sec. 3.2 on page 27) into DOMAIN, FUNCTIONAL, and NON-FUNCTIONAL requirements. The categories do have some overlap, so the individual placing can be argued. Likewise, the order in the lists are not important, it only shows some chronological discovery order.

The first list to present are related to the domain of the project. The natural context this project is used in, does imply some features and restrictions, like:

Req#	*Description* / *Reasoning*	*Category* / *Priority* / *Workload*
A1	* The actual netfilter-firewall-framework established within the Linux kernel is to be used as is. Some minor extensions or modifications may occur, but it will be kept to a bare minimum. / The kernel's framework is quite competent, allowing very capable and versatile firewall-types and -solutions to be produced. It also reduces the risks of injecting bugs into the kernel-code, and finally, it is the world the users live in - this project cannot re-invent a parallel solution.	Domain / Essential / Easy
A2	* The solution will only handle IPv4-configuration, not IPv6. / The reasons are multiple, but for instance, the new possibilities of IPv6 does not have a one-to-one mapping with IPv4. It becomes unknown how much and exactly how to handle these new possibilities. Also, IPv6 is relatively un-used by most users, and only partly implemented and supported by the current kernels.	Functional / Essential / Easy
A3	* We aim to produce an independent GUI-layer on to of an existing firewall-framework, where the base is extensible and flexible. Then the GUI-configuration-framework must be equally flexible, extensible and transparent. / That implies multiple customisable and replaceable modules, along with clear-cut APIs within the framework. Additionally, releasing and testing the implementation should improve with this general property - it's good programming.	Non-functional / Highly Desirable / Unknown
A4	* Our solution is configuring and interacting with the firewall-software within the kernel, but it isn't *part* of that software, and hence will not be released along with it. / Package-wise it is an independent release, so the framework must as far as possible, try to be equally independent on the binary level.	Domain / Highly Desirable / Unknown

Next comes the (mostly) functional requirements - and since this is a GUI-configuration-tool, the those requirements revolves around usability and interaction.

4.1.1 Usability-requirements

Usability-issues are mostly about how the user perceives and uses the solution. An overall set of headlines gave birth to the listed usability-requirements in this section. These headlines are:

Overview: The solution should present a GUI, which enables the user to see the current configuration and the processing done by the tool.
It manifest itself as req# B1, B2, B3 and B4.

Configuration: Whatever the presentation-type, it should present an clear overview, which enable the user to make adjustments in what is being presented.
It manifest itself as req# B1, B2 and B6.

Trust: The user must have confidence in the changes performed by the solution, and such confidence should be high due to the high security impact of mis-configuration. Being able to

see and change at various layers and APIs enables trust in the resulting processing.
It manifest itself as req# B3, B4 and B5.

Maintenance: The user must be able to use the tool to maintain and alter setups as time goes by. This implies a usage-cycle which reads the setup, alters it and saves it again. Also, such enables progressing of experience-levels from setup to setup - and user to user.
It manifest itself as req# B6 and B7.

With the above headlines in mind - a solution should incorporate some of the following specific requirements:

Req#	*Description* / *Reasoning*	*Category* / *Priority* / *Workload*
B1	* See the current running processes on the host, that have opened network-communication-sockets. I.e. server- and clients-processes that are poised for communication. In essence, this is specifying Input- and Output-policies for processes running locally on the host. * See the current network-interfaces and -routing on the host. Possibly with display of network-segments and -hosts found on neighbouring segment. This is specifying Forwarding- and NAT'ing-policies[1] for processes running on remote hosts parsing though the local host. ——— The above ability to see the communication-poised processes together with the network-interfaces forms the basis for firewall-rules - as experienced by the user. The user is concerned with the Question: "*What program on what computer, is to be allowed in or out?*" But, firewall-rules needs programs and computers represented as port- and IP-numbers. The port-number can be traced to the program-processes, and the IP-addresses are assigned to the network-interfaces. Hence the need for displaying processes and interfaces.	FUNCTIONAL / ESSENTIAL / EASY
B2	* See and configure the firewall's local filtering setup (Input-/Output-filters, see Section 5.1). This is specifying rules for local processes. * See and configure the firewall's forwarding setup (Forward- and NAT-filters, see Section 5.1 etc.). This is specifying rules for remote processes on remote machines (forwarding) and for rules relating to re-routing (NAT'ing). ——— Display and configuration of the firewall setup should be intuitive point-'n-click (or drag-n'-drop) operations. E.g. the user selects a network-interface (#B1), drags and drop it onto a local web-server-process (#B1), then a dialog pops up with suggested rules for allowing incoming traffic to the web-server running on the host. Had the user done the reverse, i.e. selected the server and dropped it on the interface - the rules would indicate outgoing traffic etc.	FUNCTIONAL / ESSENTIAL / HARD

Req#	*Description* / *Reasoning*	*Category* / *Priority* / *Workload*
B3	* Any point-'n-click (or drag-n'-drop) operations should be reasonable simple, and whenever confirmations or possible decisions are to be made, a dialog or wizard should guide the user. / Since some operations are difficult to fully automate, a semi-automated approach is necessary - as argued for in Sec. 2.4. But, the core goal is still to achieve as automated a process as possible, and wizards and dialogs are good for creating checkpoint-states in any process-flow.	FUNCTIONAL / HIGHLY DESIRABLE / UNKNOWN
B4	* Any changes and results should be pre-viewable before they actually take effect. I.e. the ability to see generated rules before execution, output of actual execution and result of actually accepted commands by the firewall. / Allowing inspection and confirmation of important security choices are essential. It creates a built-in solution for verifying expected results - by expert eyes. And for providing insight and teach context to less experienced users. A pre-viewable, complete, round-trip firewall-environment could be achieved by operating in a inactive/disabled part of the firewall-rules, thereby manipulating and inspecting the result and only then, actually activating the part.	FUNCTIONAL / HIGHLY DESIRABLE / UNKNOWN
B5	* Logs of execution should be saved for tracing changes and inspecting execution-trace. / Such are confidence building measures allowing back-tracking, error-traceability and forensic-investigations.	FUNCTIONAL / DESIRABLE / UNKNOWN
B6	* Ability of importing the current firewall setup (rules) and re-exporting them again, making the solution capable of performing the complete round-trip-engineering of a firewall-maintenance-cycle. / With the ability of import/export, pre-defined setups can be provided to novice users, and customised rule-sets can be carried on from host to host. In general, this is raising the overall protection-level by providing means for expert-knowledge to proliferate - just like libraries.	FUNCTIONAL / ESSENTIAL / MEDIUM
B7	* Learn new types of rules from the user, i.e. rules that aren't part of a pre-defined/-programmed setup. This involve creating a engine which can turn newly discovered rules into drag-able objects for future reference and use. / Providing means for teaching the application new abilities by scripting new rules into sets of specific tasks e.g. an anti-spoofing rule-set. An another way for expert-knowledge to be manufactured. For the tools to remain vibrant, libraries should easily created (#B7) and published (#B6).	FUNCTIONAL / DESIRABLE / HARD

Usability is a notoriously error-prone and subjective area, where good guidelines exists - but no right answers. In general, all the usability-requirement below boils down to: "I want to see it - click and change it - verify it, and thereby trust the result."

Therefore, the GUI should show the firewall-setup including the processes and network-interfaces in a clearly perceived overview. The firewall inside the kernel is used by this project *as is*, and it already have an matured conceptual model for showing packets-flows. This architectural model will be the link from processes to network-interfaces and -routing, thereby making it possible to show a complete overview of packet-flows policies for the local host. The challenge is turning the complete overview into a *clearly perceived* overview - that is clearly an usability-issue.

4.1.2 Interactiveness-requirements

The user-interactive parts should provide the flexibility, usability and feel of personal firewall-programs known from the Windows-platform. In order to do such, req# C1 must be developed, since no such functionality exists today on the Linux-platform.

A spinoff-effect of such a runtime control-link would be: Quantitative measuring by showing statistics. And putting quantity into the picture could raise a need for controlling these statistics too - using some of the existing forms of bandwidth-control (accounts for req# C2, C3 and C4).

Req#	*Description* / *Reasoning*	*Category* / *Priority* / *Workload*
C1	* In order to facilitate these more user-interactive parts, some extentions[2] to must be developed. These extensions establishes a control-link between the firewall inside the kernel and the user-space GUI-program. Such enables the GUI to pop-up dialogs for controlling the packet-flow at runtime. / A control-link enables some form of capturing network-packets that would otherwise pass though all defined rules in a firewall. They usually winds up being discarded by a default drop-all-rule as a last resort. These packets usually stems from programs for which no current rules exists. And that is the core-problem faced by many users: ”Why doesn’t this program work and how do I get to work ?”. By capturing these fall-thought, the answer would be: ”The firewall have halted the program as it was about to communicate - do you want to...” and then present the user with solutions to the communication-halt. This would provide the user with a) reasons and b) solutions - all adding up to that personal interactive touch and feel.	FUNCTIONAL / ESSENTIAL / MEDIUM
C2	* Statistics showing which ports, protocols and hosts are communication (and how much). / Measuring is the first step in tracing problems.	FUNCTIONAL / DESIRABLE / EASY
C3	* Bandwidth-throttling (also known as Traffic Control) is a way of implementing Quality-of-Service (QoS) for IPv4. According to user-preferences, certain types of traffic would gain priority over other types, e.g. interactive traffic (HTTP, SSH, etc.) over background traffic (mail, ftp, news, etc.). / Resolving performance bottlenecks and ensuring priority communication overrides inferior communications.	FUNCTIONAL / DESIRABLE / MEDIUM
C4	* Packet-Quota. Not only the type of traffic, but also the user and host involved in the communication-stream can be throttled. Such would provide QoS per user and/or host on the network. / More elaborate enforcement of the access-priority of the network-communication and -communicators.	FUNCTIONAL / DESIRABLE / HARD

Req#	*Description* / *Reasoning*	*Category* / *Priority* / *Workload*
C5	* Can we catch and stall packets from programs on the host, then it is possible to further investigate the packet contents for 'correctness'. / A re-routing of packets to a user-space program allows inspections of packet-contents, in effect adding application gateway/proxy-ing capability to the firewall. This is what e.g. the Snort-inline-project is capable of.	FUNCTIONAL / DESIRABLE / UNKNOWN
C6	* Can we catch and stall packets from programs on the host, then it is possible to verify that the programs are indeed what they claim to be. / For verification of (program) identity. I.e. by checking the integrity of the program, it is possible to stop infected or unpatched program by checking the binaries of the executing program. E.g. a check could be a simple checksum-calculation and -lookup or it could be querying a dedicated host-integrity application (like *tripwire, AIDE,* etc.).	FUNCTIONAL / DESIRABLE / UNKNOWN

4.1.3 Framework-requirements

In the more technical genre, some interface-issues have been discovered. They have been reflected in the following requirements: Req# A3 and A4.

The compile- and release-deployment (Req# A3, A4) suggests usage of runtime-linkage of the GUI towards any possible kernel-parts. Within Qt, this is referred to as "Dynamic Dialogs", and KDE mostly refers to this scheme as "KParts/KPlugins/KCMs/...".

It is a scheme where GUI-components are created at runtime from a XML-specification (plain text-file). The XML-specification contains information about what e.g. a particular dialog should contain of widgets and gadgets, along with their placement, callback-functions etc. In particular, the callback-functions become housed within their own independent object-file and isn't linked with or into the GUI-libraries (of the main-application) at all.

<u>And this is the important feature</u>: It allows extention-modules from third parties to supply an independently built control-object (call-backs) along with an XML-file (text) - and voila - the main GUI-framework can call and use the code without re-builds and -releases (hopefully).

Additionally, other issues (Req# C5 and C6) have a more futuristic use. If we are able to provide a more general interface from the kernel-space to user-space-tools though our interactive control-links (see previous requirement in section 4.1.2), then a new set of opportunities arise.

Inspecting packet-contents allow verification of the identity of program-types, i.e. verifying that some traffic on port 80 indeed are having HTTP-data in the packets.

Inspecting packet origin allow verification of the identity of program-executables, i.e verifying that some traffic on port 80 are from a particular program e.g. 'netscape v4.71 - patch 123' - verified by checksum of the executable, so that an infected or un-patched executable cannot communicate.

Such check on the communicating program can be taken to a network-wide level by using of remote lookups. Remote hosts can be queried for this type of information, and their communications can be blocked or discarded. E.g. by use of SSH or by installing customised agents in the remote hosts, the firewall can implement the actual remote query. This is in essence what e.g. SnortSAM does, as opposed to e.g. Snort-inline, which only works on the individual host.

Chapter 5

Linux Firewall-solutions

Let us sketch the up the court: What solutions are currently available? What type of firewall-structure do we have? and what firewall-solution do we want?

We have the current firewall framework present in recent Linux kernels[1], and its called *netfilter*. This is what we have to work with. And for our needs, it is quite sufficient for making solid, adequate and flexible firewalls. It is the software tool this project aims to control more easily, and we will present it shortly in Section 5.1.

However, we want the control of the packet filter to be more easy and flexible. The current control of the filter is basically a command-line interface, where the commands are usually executed by a startup-script. Once installed it is usually a fairly static setup, and it's not easy to handle and requires computer-skills beyond most users. The new desired layout (for controlling netfilter) will be outlined in the following Chapter 6 on page 47.

5.1 The Linux packet filter (netfilter/iptables)

Our tool for building the actual firewalls on Linux is named the *netfilter*-package. Technically, it is a packet-filter-firewall, meaning it is working on the packet-headers[2] and not the packet-contents[3]. The netfilter does have extendibility to accommodate filtering beyond looking at only the headers, but such is mostly outside of netfilter-modules, and is reached by third-party software though the netfilter API. This can turn the packet-filter firewall into a *application-level gateway*, which have knowledge about a specific application and can allow connections based on packet-contents. E.g. *snort* (nicknamed snort-inline) can be hooked up on netfilter's QUEUE-target[4], analyse and issue accept-/drop-verdicts of the packets on the queue.

Sometimes the term *proxy* is used for such, and this is an overloaded term of the application-level firewall. A proxy takes the application-knowledge even further by taking over the communication on behalf of the client - essentially cutting the line and and act as an controlling middle-man - not just allowing the communication with the client.

The netfilter-module resides inside the kernel and filter network-packets according to some pre-loaded rules. The administration program to control netfilter is named *iptables* and it is a command-line program. Essentially, one never sees netfilter - its inside the kernel - but one uses iptables whenever the firewall is managed, hence the "Linux firewall packet-filter" is generally referred by the name *iptables*, though strictly speaking, that is just a program-file in the *netfilter*-package.

[1] for kernels V2.4 and onwards.

[2] working on the TCP/IP-headers - or 3'rd and 4'th level of the OSI-7 model.

[3] application layers - or 5'th to 7'th layer of the OSI-7 model.

[4] The QUEUE-target send a matched packet to the userspace-program hooked up to the queue - for more, see the explanations later in this section.
We'll use the QUEUE-target too for our packet capture.

The rules of netfilter are organised in tables called *chains* (i. e. lists). Each rule consists of a *matching* section and a *target* section. If a packet is matched, it is sent to the target and the packet is considered processed. Several match-types can be specified in a section and many match-types exists: E.g. for matching on protocol (TCP, UDP, IGMP, RTP, IPSec. . .), IP-address or port (Source-/Destination-), or for packet-length, -contents, -mark (see *mangle*-chains later). The target determines the faith of the packets matched in this rule. The target can be simple actions (Accept, Drop, Reject, Log, Mark), or it can refer the packets to further processing in a user-defined chain (sub-list of further rules), it can alter packets (NAT'ing/Mangling), or it can pass packets on to a user-space program for inspection and more advanced customised processing (Queue).

An example can be seen in Appendix A on page 93.

Filtering The user are to control links between processes (programs) and network-cards (NICs). This is the core of what firewall-rules are to archive. This project's GUI are to handle an iptables-firewall, and therefore the GUI must present the processes and the NICs - as seen in the context of the iptables paradigm. The iptables paradigm is essentially how a network packet flows through the Linux-box - See Fig 5.1.

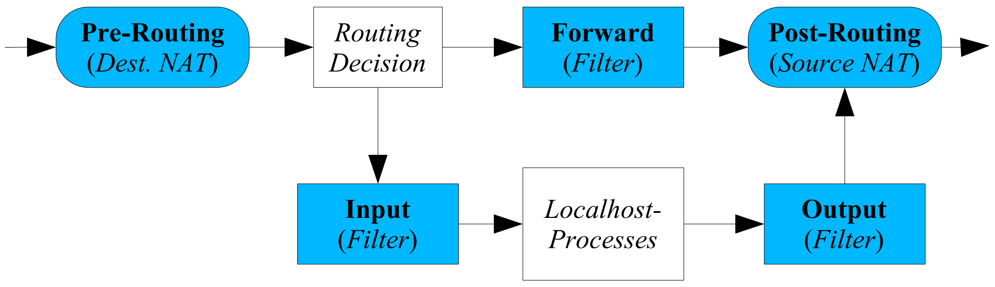


Figure 5.1: Netfilter-flow (iptables) -(See also Fig. 6.5 on page 57 for a more complete view).

Filtering takes place in various chains, depending on the source of the packets. Packets comes into the chart from two possible sources:

1. Received on some network interface (e.g. some ethernet-card, serial-, wireless- or infrared-device etc.) - represented by the arrow on the far left. In this case, a routing decision is made: **either** the packet is for this host and is sent to a local process (through the Input-chain), **or** the packet is to be sent to some other host (through the Forward-chain).

2. Received from a local process running on this host (locally generated packet - lower-centre-box). In this second case, the packet is to be sent out on the network (through the Output-chain), using the appropriate network interface (NIC) for the packet's destination.

The core firewall functionality is in the three filter-chains. This is where the rules for accepting and denying is placed. The Input- and Output-chains contains filter-rules regarding packets for the local host only. If the packet isn't for this host, it is forwarded (filtered by the Forward-chain) to the proper interface (network segment) - or to the default gateway if it isn't a segment connected directly to this host.

NAT'ing Most firewalls can do **N**etwork **A**ddress **T**ranslation (NAT) on the packets, at the same time as filtering. That make sense to do simultaneously, since the filter looks at the header of the packet anyway (to determine the type), and in that process, the firewall might as well do the NAT-manipulation of source- and destination-addresses of the packets. Thereby, the implementation is optimised and organised into one work-flow instead of two - but bear in mind: filtering and

NAT'ing is two distinct issues and they are only implemented together for efficiency. *In short, filtering is about access-control, NAT'ing is about re-routing.*

In Fig 5.1, the NAT-chains are the rounded boxes. Most users do Source-NAT'ing, because they are possessing only one public IP-address. Therefore they need to substitute the Source-IP-address on all outgoing packets with the public assigned IP - in order to get any reply back. A firewall with only one public address protecting a local LAN-segment of machines behind it, must do Source-NAT'ing on all packets from all machines in it's LAN-segment. A simplified variation of Source-NAT'ing is called masquerading and used on dialup-accounts where the assigned IP changes very frequently.

Destination-NAT'ing is done more infrequently. It is used when the firewall is acting as the front-end of a server-farm. It can thereby act as the single-point-of-contact to the outside world, that perceivably runs services, but in fact it re-routes traffic to an DMZ-zone with the real servers (web, ftp, mail etc.). Or it can do large scale load-balancing by constantly changing the destination address to various machines in the farm in order to distribute the e.g. web-server-load.

An application of NAT'ing is encrypted traffic (tunnelling) on the transport-level (using IPSec) is also done with use of NAT'ing. The NAT-host is an encryption-endpoint in the encrypted tunnel, and this is where header can last be seen and inspected before the packet - including the header - gets encrypted. Hosts and routers beyond this point cannot see the TCP-header and higher-level information, only the IP-header of source- and destination-IP is visible so they can pass the packet on.

Mangling A third group of chains exists within netfilter, referred to as the *mangle*-chains (Input-, Forward-, Output-, Pre- and Post-chains). There exists one mangle-chain for each box in Fig 5.1, and hence mangling exists for both filter- and NAT-boxes. A particular mangle chain is consulted right before the corresponding filter-/NAT-chain is. Mangle-rules allows for marking particular types of packets (E.g. marking outgoing SSH-packets with lengths less than 500 bytes...) and use the marker later on in some NAT- and filter-chain for some particular processing of choice.

Traffic Control (*Quality-Of-Service, QoS*) is another aspect of firewalling. When combined with filtering over time, it allows the guarantee of bandwidth-usage according to laid-out policies. By looking at the packet-headers over time, certain traffic-streams may be throttled by dropping packets from the stream when surpassing a certain volume of traffic. The stream is still allowed, but limited in volume.

Traffic-statistics have to be collected in order to do Traffic Control. The ULOG-target (in combination with ulogd-demon) can log traffic-info to a database (e.g. Postgres) from which traffic amounts can be calculated. The statistics also serve as back-logs for load-measuring, investigations and forensic-work.

Key points: The currently existing framework within the Linux kernel is what we will use. It provides many opportunities for controlling just about any aspect of packet-traversal on the network. But, aid in setting up the type of firewall, the ease and flexibility in manipulation and controlling the rules <u>isn't</u> within the domain of netfilter - and that's to be developed.

packet-filter A firewall that filters the packets based on packet-headers, operating on OSI-7-layers 3-4.

application-level-gateway A firewall that filters the packets using rules which also takes packet-contents into account. Operates on OSI-7-layers 3-4 and 5-7.

proxy-gateway A firewall that filters the packets also taking packet-contents into use, and *in addition* shunts the connections between the client and server to the proxy instead. The proxy is acting on behalf of the client, creating the only connection to the server and vice versa towards the client - becoming the man-in-the-middle. Operates on OSI-7-layers 5-7.

netfilter The software tool present in current Linux kernels for packet-filtering firewalls. It creates the framework of organising traffic

iptables The command-line program to control netfilter.

rules The rules loaded into the netfilter for allowing and denying specific network-packets.

chains The rules are organised into lists (chains) and sublists. Build-in chains are forming the routing framework of packets flowing through the filter. Custom-made chains allow more easily organisation of rules for better and faster performance. Correct name is Filter-chains.

NAT A packet manipulation, also done during packet inspection, *but it is not for access-control*; its for routing, masquerading and other remapping of IP-addresses and TCP-ports. Correct name is NAT-chains.

Mangle More general and customisable packet-manipulation-chains than NAT-chains providing more clear and clean rule-organisation. Correct name is Mangle-chains.

The development of better software for manipulating and controlling the netfilter is next. But before we sketch our solution, we will investigate the currently available solutions. This way, we avoid observed pitfalls and duplication of efforts and - generally speaking - gets inspiration and circumvent reinventing the wheel again - we'll just steal it instead!

5.2 Survey of existing OpenSource-solutions

Several tools exists on the net[5] - all found by searching for the word *firewall*, at least 25 tools have been seen. Most of the projects are in a simple or closed state of development. The majority of solutions are mostly written in BASH - and hence, they all have a distinct command-line-aroma over them. This includes the major Linux-vendors[6] and most home-grown pet-projects. All of them have their own config-scripts and possibly a get-started-wizard of some sort. But - as discussed later on - not all solutions are based on scripts and some of them bear noteworthy properties.

Some PHP-based solutions exists in the net, where users can enter some very basic information on a webpage, and a BASH-script of iptables-commands are generated and sent back to the user. Some of these implies some form of payment[7] for this 'service', and the quality isn't impressive.

In general, none of the tools found could import the iptables-rules currently loaded on a host. All rules have to specified inside the tools, which then generate the rules. Very few tools offer wizards or pre-defined typical setups - the user must start from scratch each time. Since import isn't possible, a round-trip engineering cycle is impossible - changes must come from within the tool itself.

FireHOL: Some scripts are more wizard-driven than other, especially the vendor-provided scripts, but one intriguing and different solution is *FireHOL*.

It is still written in BASH, but have a very simple language - see Table 5.1.

The simplicity of the language does not overshadow the possibilities of the language: it is very flexible indeed, without cluttering the descriptions. Looking into the language it is easy to see, that 'router' means forward-chain, 'server' means input-chain, and the output-chain is represented by 'client'. Also, symbolic names are used for most things, making the language more familiar to the local environment and more readable.

To enable further discussions two of the most interesting GUIs is presented next. First the specialised *KMyFirewall*, then the generalised *Firewall Builder*.

[5]Especially http://sourceforge.org and http://www.freshmeat.net

[6]RedHat, SuSE, Mandrake etc.

[7]The possibility of earning a few pennies in this way, shows the state of affairs for the unknowledgefull user!

FireHOL Language-example:	Notes:
`server_imap_ports="tcp/143"`	-> Defines a server-service - here the imap-mail protocoll
`client_imap_ports="default"`	-> Ditto for clients - same port Note: Many services are pre-defined (the above is just a snippit from these)
`interface ppp0 myisp`	-> Defines the first (external) interface
`server imap accept`	-> Do accept incoming from external network
`interface eth1 homenet`	-> Defines the second (internal) interface
`client imap reject`	-> Don't accept incoming from internal network
`router home2inet inface homenet outface myisp`	-> Defines a forwarding chain from internal to external (but not the other way too)
`route imap accept`	-> Do forward this traffic from internal to external network

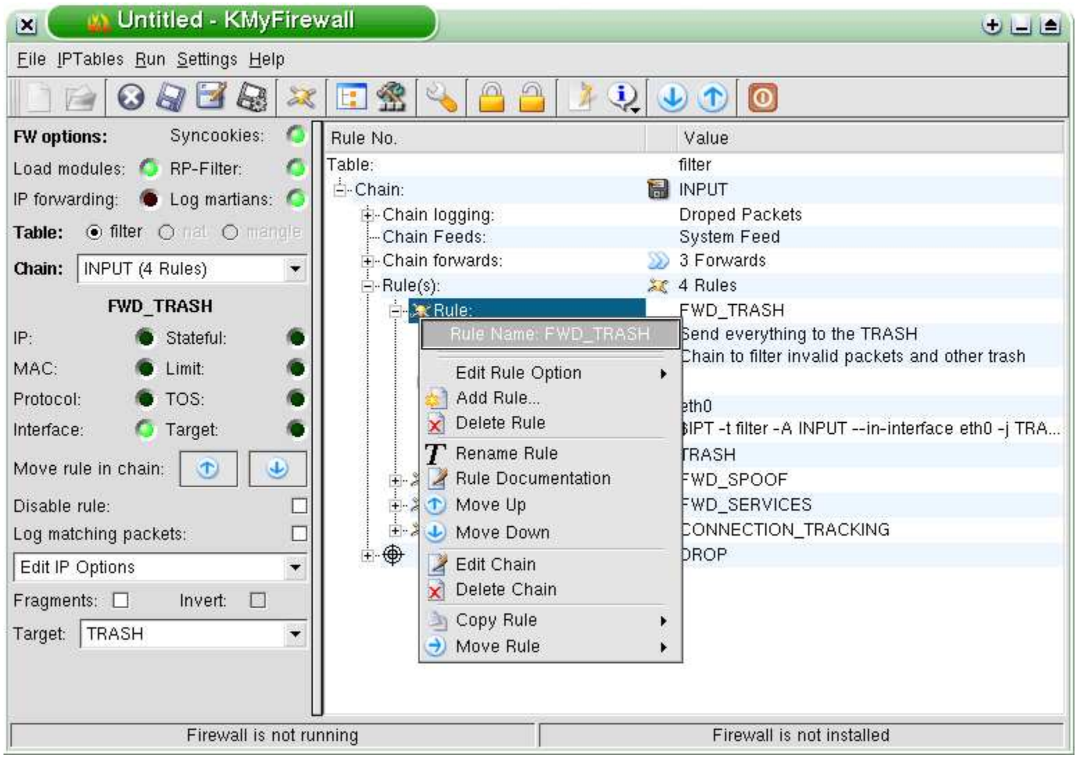
Table 5.1: FireHOL example.

KMyFirewall: This GUI is written specifically for iptables and its clearly seen - see Fig. 5.2. Following are description and Screenshot (Fig. 5.2) from the homepage:

> This is the project homepage for KMyFirewall, an IPTables based firewall configuration tool for the KDE Desktop Environment running on Linux based systems.
>
> KMyFirewall attempts to make it easier to setup IPTables based firewalls on Linux systems. It will be the right tool if you like to have a so called "Personal Firewall" running on your Linux box, but don't have the time and/or the interest to spend hours in front of the IPTables manual just to setup a Firewall that keeps the "bad" people out.
>
> There is also the possibility to save entire rule sets, so you only have to configure your rule set one time and then you can use it on several computers giving each of them a similar configuration (p.e. school networks, office, university etc.). For a complete list of the features have a look at the Features section
>
> Programs can't do any magic so you still will have to know what your firewall should do to setup your rule set. KMyFirewall just tries to help you as much as possible, but you decide what it will do.
>
> Currently we are focusing on the 1.0 release, after which the user interface will be expanded to enable more easy configuration for both advanced and novice users. Also NAT support will become more accessible to setup. In the mean time, try out the latest release!

KMyFirewall - shown in Fig. 5.2 - is a more direct implementation of a iptables-GUI. It has nice looks, but does not aid the user in any way - apart from presenting the possible flags of each command, so that the user doesn't have to read man-pages in order to find out what is possible with each rule. There is no aid on how to setup rules, and no overview is presented either. But it does offer more advanced possibilities, since more elaborate rules than simply accepting or denying is possible.

The KMyFirewall is a pure, but also a very straightforward attempt with nice graphics - it is one of the nicer layouts seen so far (*. . . but fear the rest*).



FW options: Syncookies:
Load modules: RP-Filter:
IP forwarding: Log martians:
Table: (● filter) ○ nat ○ mangle
Chain: INPUT (4 Rules)
FWD_TRASH
IP: Stateful:
MAC: Limit:
Protocol: TOS:
Interface: Target:
Move rule in chain:
Disable rule: ☐
Log matching packets: ☐
Edit IP Options
Fragments: ☐ Invert: ☐
Target: TRASH

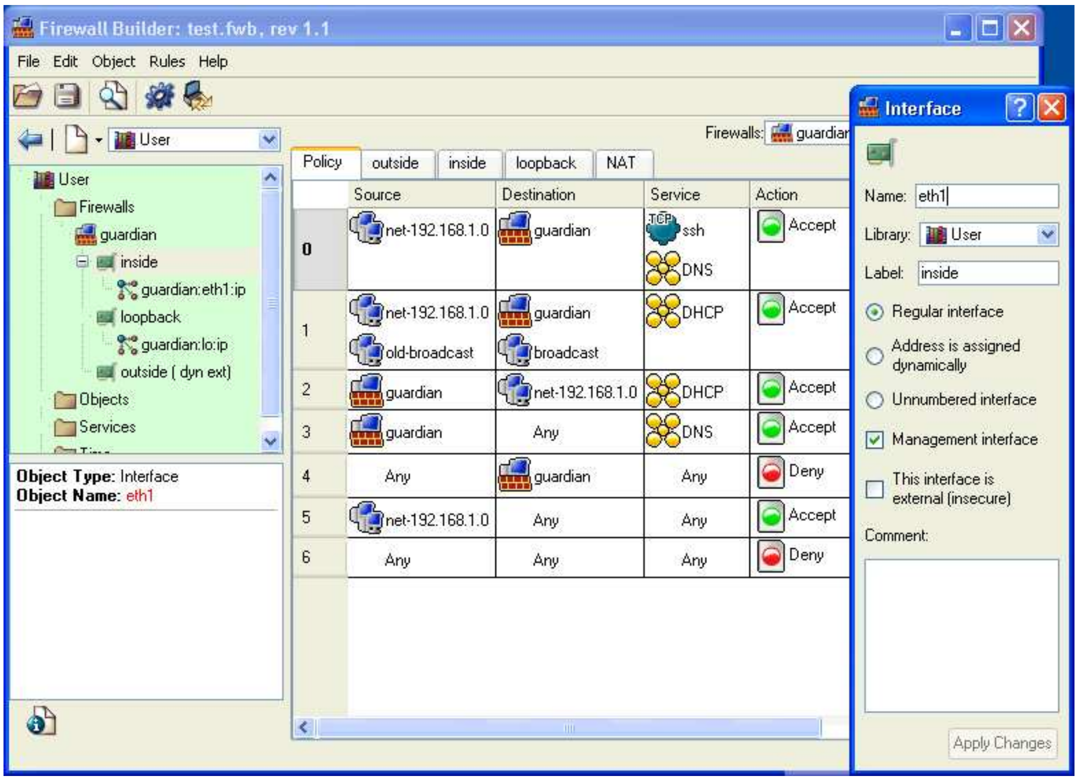
Rule No.	Value
Table:	filter
Chain:	INPUT
Chain logging:	Droped Packets
Chain Feeds:	System Feed
Chain forwards:	3 Forwards
Rule(s):	4 Rules
Rule:	FWD_TRASH
	Send everything to the TRASH
	Chain to filter invalid packets and other trash
	eth0
	$IPT -t filter -A INPUT --in-interface eth0 -j TRA...
	TRASH
	FWD_SPOOF
	FWD_SERVICES
	CONNECTION_TRACKING
	DROP

Firewall is not running | Firewall is not installed

Figure 5.2: KMyFirewall.

Firewall Builder: A much more generic and professional structured solution is Firewall Builder - shown in Fig. 5.3. Following are description and Screenshot (Fig. 5.3) from the homepage:

> Firewall Builder is multi-platform firewall configuration and management tool. It consists of a GUI and set of policy compilers for various firewall platforms. Firewall Builder uses object-oriented approach, it helps administrator maintain a database of network objects and allows policy editing using simple drag-and-drop operations. Firewall Builder currently supports iptables, ipfilter, OpenBSD PF and Cisco PIX. Technical summary of features supported by the policy compilers for all platforms can be found in the section "Modules" (see menu on the left).
>
> Being truly vendor-neutral, Firewall Builder can generate configuration file for any supported target firewall platform from the same policy created in its GUI. This provides for both consistent policy management solution for heterogeneous environments and possible migration path.
>
> Firewall Builder allows for management of multiple firewalls using the same network object database. Change made to an object is immediately reflected in the policy of all firewalls using this object. Administrator only needs to recompile and install policies on actual firewall machines.
>
> In Firewall Builder, administrator works with an abstraction of firewall policy and NAT rules; software effectively "hides" specifics of particular target firewall platform and helps administrator focus on implementation of security policy. Backend software components, or policy compilers, can deduct many parameters of policy rules using information available through network and service objects and therefore generate fairly complex code for the target firewall, thus relieving administrator from having to remember all its details and limitations. Policy compilers can also run sanity checks



Object Type: Interface
Object Name: eth1

	Source	Destination	Service	Action
0	net-192.168.1.0	guardian	ssh DNS	Accept
1	net-192.168.1.0 old-broadcast	guardian broadcast	DHCP	Accept
2	guardian	net-192.168.1.0	DHCP	Accept
3	guardian	Any	DNS	Accept
4	Any	guardian	Any	Deny
5	net-192.168.1.0	Any	Any	Accept
6	Any	Any	Any	Deny

Name: eth1
Library: User
Label: inside
[✓ Regular interface]
Address is assigned dynamically
Unnumbered interface
[✓ Management interface]
This interface is external (insecure)
Comment:

Figure 5.3: Firewall Builder.

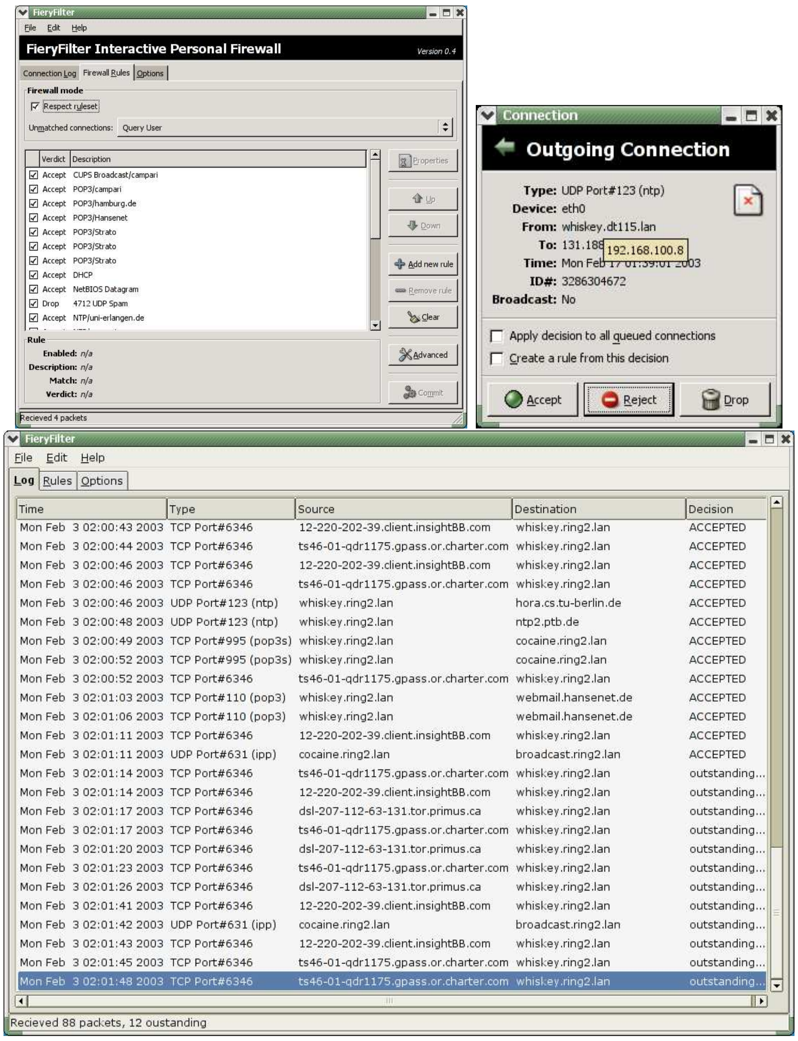
on firewall rules and make sure typical errors are caught before generated policy is deployed.

Noticeable in Firewall Builder (Fig. 5.3) is the tree-structure on the left and the list-box in the tabbed pane. The tree-structure are having one node for each service on each interface, which easily accumulates to as many lines as a raw iptables-dump. On the positive side some logical abstract grouping is done. The listbox in the tabbed pane is no more than an GUI-presentation of 'flag-slipping' of the command-line syntax. The numerous, but restrictive tabs are concealing information and is catering more for the command-line syntax than the overview. The information explosion is good. The dialog shown on the right is popped upon clicking on a interface - it gives the necessary and expected details without cluttering information in the tree-view.

The Firewall Builder is an interesting solution. It is cross-platform-ed and have abstracted rules in to an object form (represented in XML). It tries to capture the core of rule-building and not focus on the details of a firewall on this or that platform. It operates on multiple firewalls from one database, maintaining an overall perspective of the network - not just the individual firewall-host. But unfortunately, it seems to offer the lowest common denominator of all the firewall-platforms it supports: simple accept and deny on hosts and port numbers. Also, the GUI isn't presenting an integrated view-model, it more or less presents nice click-able lists of raw table-dumps - the user must independently maintain an overview of the network and policies - mostly unaided by the tool.

FieryFilter: This program - shown in Fig. 5.4 - was discovered as this project was uploaded to freshmeat[8] at the end of the project period. It turns out that another have had the same idea of capturing packets and popup a dialog requesting the user to accept or deny the connection. It

[8]http://www.freshmeat.net



Firewall mode
[✓] Respect ruleset
Unmatched connections: Query User

	Verdict	Description
✓	Accept	CUPS Broadcast/campari
✓	Accept	POP3/campari
✓	Accept	POP3/hamburg.de
✓	Accept	POP3/Hansenet
✓	Accept	POP3/Strato
✓	Accept	POP3/Strato
✓	Accept	POP3/Strato
✓	Accept	DHCP
✓	Accept	NetBIOS Datagram
✓	Drop	4712 UDP Spam
✓	Accept	NTP/uni-erlangen.de

Rule
Enabled: n/a
Description: n/a
Match: n/a
Verdict: n/a

Recieved 4 packets

Type: UDP Port#123 (ntp)
Device: eth0
From: whiskey.dt115.lan
To: 131.188 192.168.100.8
Time: Mon Feb 17 01:59:01 2003
ID#: 3286304672
Broadcast: No

[ ] Apply decision to all queued connections
[ ] Create a rule from this decision

Time	Type	Source	Destination	Decision
Mon Feb 3 02:00:43 2003	TCP Port#6346	12-220-202-39.client.insightBB.com	whiskey.ring2.lan	ACCEPTED
Mon Feb 3 02:00:44 2003	TCP Port#6346	ts46-01-qdr1175.gpass.or.charter.com	whiskey.ring2.lan	ACCEPTED
Mon Feb 3 02:00:46 2003	TCP Port#6346	12-220-202-39.client.insightBB.com	whiskey.ring2.lan	ACCEPTED
Mon Feb 3 02:00:46 2003	TCP Port#6346	ts46-01-qdr1175.gpass.or.charter.com	whiskey.ring2.lan	ACCEPTED
Mon Feb 3 02:00:46 2003	UDP Port#123 (ntp)	whiskey.ring2.lan	hora.cs.tu-berlin.de	ACCEPTED
Mon Feb 3 02:00:48 2003	UDP Port#123 (ntp)	whiskey.ring2.lan	ntp2.ptb.de	ACCEPTED
Mon Feb 3 02:00:49 2003	TCP Port#995 (pop3s)	whiskey.ring2.lan	cocaine.ring2.lan	ACCEPTED
Mon Feb 3 02:00:52 2003	TCP Port#995 (pop3s)	whiskey.ring2.lan	cocaine.ring2.lan	ACCEPTED
Mon Feb 3 02:00:52 2003	TCP Port#6346	ts46-01-qdr1175.gpass.or.charter.com	whiskey.ring2.lan	ACCEPTED
Mon Feb 3 02:01:03 2003	TCP Port#110 (pop3)	whiskey.ring2.lan	webmail.hansenet.de	ACCEPTED
Mon Feb 3 02:01:06 2003	TCP Port#110 (pop3)	whiskey.ring2.lan	webmail.hansenet.de	ACCEPTED
Mon Feb 3 02:01:11 2003	TCP Port#6346	12-220-202-39.client.insightBB.com	whiskey.ring2.lan	ACCEPTED
Mon Feb 3 02:01:11 2003	UDP Port#631 (ipp)	cocaine.ring2.lan	broadcast.ring2.lan	ACCEPTED
Mon Feb 3 02:01:14 2003	TCP Port#6346	ts46-01-qdr1175.gpass.or.charter.com	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:14 2003	TCP Port#6346	12-220-202-39.client.insightBB.com	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:17 2003	TCP Port#6346	dsl-207-112-63-131.tor.primus.ca	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:17 2003	TCP Port#6346	ts46-01-qdr1175.gpass.or.charter.com	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:20 2003	TCP Port#6346	dsl-207-112-63-131.tor.primus.ca	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:23 2003	TCP Port#6346	ts46-01-qdr1175.gpass.or.charter.com	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:26 2003	TCP Port#6346	dsl-207-112-63-131.tor.primus.ca	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:41 2003	TCP Port#6346	12-220-202-39.client.insightBB.com	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:42 2003	UDP Port#631 (ipp)	cocaine.ring2.lan	broadcast.ring2.lan	outstanding...
Mon Feb 3 02:01:43 2003	TCP Port#6346	12-220-202-39.client.insightBB.com	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:45 2003	TCP Port#6346	ts46-01-qdr1175.gpass.or.charter.com	whiskey.ring2.lan	outstanding...
Mon Feb 3 02:01:48 2003	TCP Port#6346	ts46-01-qdr1175.gpass.or.charter.com	whiskey.ring2.lan	outstanding...

Recieved 88 packets, 12 oustanding

Figure 5.4: FieryFilter.

also uses the QUEUE-target for capturing the packets. It's intended functionality are very similar to our project-requirements for the personal, interactive and dynamic aspect of the firewall.

Following are description and Screenshot (Fig. 5.4) from the homepage:

> FieryFilter is an interactive desktop firewall for Linux. It will ask the user every time a new network connection is made if they want to allow or deny it. The user is able to generate rules from connections and thus minimise the amount of questions asked.

But, FieryFilter does not allow any other configuration of the firewall than it's own - it wipes the firewall of any existing rules and deploys a fixed setup, to which it knows where to find insertions points for permanent connection-decisions.

Neither can it make any link to what program or process is involved (getting/originating) - only protocol, source- and destination-IP and -ports are shown.

It operates on a packet-by-packet base, and not per connection - i.e. a SSH-client will send several packets, both while waiting for ACCEPT'ance and while running the remote shell once admitted into the host. It does not talk to iptables, other saving the previous firewall-rules to a file before wiping.

It does not seem to be active any more and the last update was V0.4 in 2003.

Summery: Unfortunately, the fourth solution - *FieryFilter* - was discovered way too late to have any impact on the project. But it would have resolved some initial investigations into how to perform packet-capturing, and what type of data to expect and sketch the interface, that would be available and needed. When skimming though the source code of FieryFilter, it has very recognisable structures, which pretty closely mimics the basic data-structures in the equivalent parts of our solution.

The first three highlighted solutions (*FireHOL*, *KMyFirewall* and *Firewall Builder*) serves as inspiration:

- It should complete the roundtrip engineering cycle of building and maintaining firewall-rules. Therfore it must be able import the currently loaded rules from iptables, manipulate and store the rules back into the firewall using iptables - no other solution does.
- Our solution should be modelled by the same structure implied by FireHOL. That is, the general point of view, is to map any action against the three main chains of netfilter: Input, Output and Forward. This model of the FireHOL specification language will the model of our choice. Indeed, maybe it will be feasible to use FireHOL behind our GUI as part of the implementation language generating the actual iptables-commands - it captures the internal structure of netfilter (as explained in Sec. 5.1) very well and to the point.
- It should be possible to utilise the more intricate details of netfilter modules, providing GUI-dialog-support for all possible specialised flags and switches of netfilter built-in- and extension-modules - as KMyFirewall does.
- It would be desirable to take the holistic, network-wide, platform-independent and object-oriented view of Firewall Builder. It has some nice domain-modelling thoughts in its design - i.e. making it network-wide with the possibility to operate on/with hosts behind the firewall.
- Usability is important. Easy usage allows for more customised and precise rules - if it isn't easy *to* use, it won't *get* used!

Our solution will lend, borrow and steal any desirable feature found in these solutions.

5.3 Related academic work

Some work have been made in ensuring that the firewall-rules adhere more closely to the intended security-policies. The research have been focusing A) detecting errors and B) preventing the errors.

The work of detecting errors in the rules, have been focusing on trying to construct automatic rule-checkers, verifiers etc, that could alert the users by detecting (and possibly correcting) the errors - or at least detect anomalies in the rules and warn the users.

Others have focused on the prevention of errors in the first place - by elevating the specification level from (low-level) firewall-rules to (higher-level) security-specifications. Thereby, the rules are controlled by the specification-language and therefore hidden from direct manipulation. The trick then is to find the balance between flexibility in the languages to express the needed diversity in filtering and monitoring of traffic - and the ability to ensure the language will never reach an undesired state, and thereby rendering erroneous-rules.

Two recent pieces which seem to stand out are: *fang[Fang]* and *firmato*[Firmato]. They are referenced by many articles and seem focus on the two central themes described above.

The Fang Rule-analyser Fang is analysing rules to provide answers of '*. . . is there access from where-to-where (host/zone) with what-protocol (service). . .* '. It should fit right in our application-frame as a part - with few alterations, since it's already a Qt-application.

It uses a description[9] of the network-layout along with the configuration-files[10] of the firewalls and gateways in the net. By analysing the configurations and knowing the network-layout, it can assess the query of '*what hosts can reach where, through what ports*'.

In order to use it in our context, we need to convert our setup-wizard's NIC's-to-Zones-format, into the MDL-description-language of fang. Also, the iptables-format must be understood by fang, so another fang-configuration-file-front-end must be made (like for Cisco's IOS-format). The capabilities of iptables-rules by far supersedes the possibilities of fang, and is unresolved how to handle iptables-rules which aren't simple actions of accept/drop of packets to/from ip-addr/-port. But if rules are sufficiently clean enough, fang should be able to function within our frame-work (with some patch-work).

A new tool called *ITVal[ITVal]* does what fang does, but for iptables specifically. It parses iptables-output and allows queries of '*what hosts can reach where, through what ports*', but only for the current firewall and without NAT'ing. It does not have a Qt-GUI, it uses a commandline-interface. It should be portable, but the exact details haven't been scrutinised.

The Firmato management tool/language Firmato is a management tool which deals with a more high-level description of the security-policy, than mere rules and their sequences. It tries to e.g. be what C-language is to CPU-architectures, compared to using assembler-language. You'll need to re-write the security-policies for every specific device, if you use low-level rules (e.g. iptables or cisco IOS), instead of inventing a language (i.e. *firmato*-MDL) and a compiler for generating the configurations.

A snip from the article's abstract:

> In this paper we present Firmato, a firewall management toolkit, with the following distinguishing properties and components: (1) an entity- relationship model containing, in a unified form, global knowledge of the security policy and of the network topology; (2) a model definition language, which we use as an interface to define an instance of the entity-relationship model; (3) a model compiler, translating the global knowledge of the model into firewall-specific configuration files; and (4) a graphical firewall rule illustrator.

Well, as can seen later in Sec. 6 on page 47, it sounds very much along the lines of our solution - though not quite with the same purpose in mind.

Firstly (1), we also have a database with ER-relations of rules, but for the purpose of unified, flexible, independent API-access to rules - not as an abstraction of rules into a new elevated model. Secondly (2), we lack a modelling language (*firmato*-MDL) - we use iptables as our 'language', since we have an direct link between our ER and our 'modelling' of rules.

Thirdly (3), our compiler is fairly simple in comparison, it's a conversion more than a compiler, since we don't really have language (apart from iptables). In contrast, we don't just generate from or database, but also parse the complied output back in for round-trip modifications. Forth (4) and finally: Yes we will strive to make graphical illustrations of our rules and processes. Unfortunately, we don't make it to item (4) due to time-constraints and lack of ideas on how to handle diversity in the rules of iptables (we got hit by real-life-usage. . .).

To use firmato in our context, our framework does cater for it by having a SQL-database for it's ER-model to reside in. The MDL-language is the core of their concept and can be used as is. The model compiler needs to be extended to generate iptables-format too, but that shouldn't pose a major problem, since it's a one-way street (firmato-to-iptables) and iptables can encompass all rule-compositions required by firmato. The (4) graphical illustrator requires some work. Both to port/patch into the Qt-frame - and in functionality and design as described in the conclusion of their paper.

[9]The description-language is written using a subset of the *firmato*-MDL-language.

[10]The rules of each supported brand of firewall - e.g. currently supported front-ends are: Cisco Routers IOS and Lucent firewall. Each specific firewall has it's own device-dependent way of specifying rules, and has various filtering capabilities.

Indeed, many of the concluded issues of their paper are issues this project can sign on too, as far as the rule-modelling, -manipulation and -presentation is concerned.

Summary All the academic papaers are fairly advanced work, that may (- or may not ;-) work in practice. They all seem to use logic programming (Prolog/Eclipse) and work with packet-filters only - i.e. they only consider the closed world assumption[11] of looking (matching) at a packet's *source* and *destination* of IP-*addresses* and TCP-*ports*, with verdicts (targets) of either *Accept* or *Drop*.

Note, I don't think that assumption does holds up in real life, because:

A) The Linux-netfilter have <u>33</u> types of matches and <u>22</u> types of targets - and that's just in the standard package, many more are available by *patch-O-matic*[12].

B) Hackers are not nice and will not comply with well-formed packets on expected ports and ip-addresses, so a packet's origin interface (NIC on OSI-layer 2) must be considered too, in order to avoid simple spoofing, re-routing, or broadcast-attempts to circumvent the packet-filter. Also, the statefull-ness of the filter must be addressed. Is it a new, established or related connection? - otherwise the TCP-SYN, -ACK, and other flags can circumvent the filter by implying that the packet is belonging to an ongoing connection.

C) Real-life network-threads are also including packet-contents, and that opens up the necessity to know e.g. snort's entire possible decision-tree in order to include it into the closed world assumption of the packet-filter. After all, if we don't know what verdict e.g. *snort* will cast (Accept or Drop) on a packet - what can we claim to know about that rule? - how will it handle a packet?
In general, any packet can be sent to a user-space-program, that can cast verdicts. But it is unspecified by these tools, what packets will go there, what the verdicts will/can be, and whether or not a packet will be identical (not mangled) coming back to the firewall-filter. Any user-space-program must be know by these tools in order to let them consider the possible outcome.

D) Most of them have their own higher-level specification language, instead of messing about with the rules and the order of the rules. They then use compilers to form rules from their high-level specification language.
But most of them, *do not* allow fiddling behind the tool's back with the rules directly (which we can hardly blame 'em for). Reality is though, that real-world-solutions have difficulties in shunting out such possibility - the *need* to fiddle is still present, despite these solutions.
In this aspect, they are just like the existing OpenSource-solutions: They wipe the firewall of rules and will only insert rules which *they* know about and are familiar with, in *their* expected and predicted order. They only recognise issues in their closed-world-assumption, and that means no use of 3.party extensions or 3. party programs for handling issues they *don't* know about - and that's the hole point in extensions...

E) Logic programmning is hard! Some of the work include descriping the high-level security-policies in logical programming-languages. Most system-administrators don't know how to express firewall-policies in logical phrases for such a language (E.g. Prolog, Eclipes etc.) - and neither does many programmers (although some argue they should). That leaves researchers to either finish the job themselves[13] and completely show that *this hammer does hit the nail.* Or to completely hide the logical inference engine and it's knowledge-database deep underneath the hood in a tool - and sell the tool instead.
Because maintenance-wise, somebody with logic programming skills must be present in each

[11]Finite fallout-space, hence all possible scenarios can be considered.
[12]See http://www.netfilter.org for more information on applying the 3.party contributed extensions to the kernel, by using patch-O-matic.
[13]It's not sufficient only to show proof-of-priciple, ie. it *cannot* be left as 'an exercise for the reader to finish'...

computer-administration-department (and in the home-user's private household).
I predict - that if managers have to re-trained or schooled in Prolog to do firewall-setup and -maintenance - then most of them will find other means to get the job done (or simply find new careers).

Warning - my personal opinion: Don't get me wrong - I really *do think highly* of the papers above and the work put into the research area - but personally, I think the logical programming approach is a bit of a dead-end for now. It's a good idea and probably very close to how policies are formulated in words - it just isn't finished and usable yet. Furthermore, it is reflected in the fact, that most of them are still in prototype-stages, where a working proto-type (or proof-of-principle) have been made - but none of them are really in massive deployment in real life[14] (*...end of warning*).

More on the issue in Sec. 8.1 on page 84.

Our solution will try and embrace whatever they bring to the party - by allowing the tools to exists as modules within our frame (by patching them). They do offer the best attempts on providing more automated and/or elevated security-specification by using higher-level-languages. And if possible, we would try and provide a frame for their attempts to develop and deploy within.

Likewise, our automation-attempts would be much smaller, shorter, more naive and of less extend and quality, than theirs. So, we will try and leverage the combined solution by making a modularised framework-structure, something most other existing OpenSource-solutions haven't catered for at all.

[14]Note: that doesn't exclude buisness-opportunities to make and sell products built on this approach.

Chapter 6

Our solution - User interface and System Architecture

The main focus of the GUI is to compose a clever presentation of the desired firewall-rules. Of course, compressing a lot of information into a simpler view have it's own problems - i.e. of what and how to present the selected information. Several modelling views can be chosen as the 'right' one. In the following, we will sketch the end-objective of our solution.

Examples of already existing models and views where presented in Sec 5.2. The intention is to avoid the very detailed layout and 'flag-switching'-nature of these GUI'es. But it *must not* exclude the option of fiddling with such detailed flags - it just shouldn't make up the entire approach and view port of the firewall-model. The existing solutions are very true to the nature of the command-line they wrap around - and they reflect the commands well, but they don't take an elevated view of the domain. Most of them (in essence) presents an set of GUI-dialogs for the command-line programs.

When it comes to selecting a good view, several arguments can be put forward. For instance, it should reflect the actual technology underneath. And it should speak the end-user's language. Fine - both are good arguments... and here's a story from real life, where both arguments where catered for (more or less) - and it still went wrong!

> In my former youth as an apprentice (back in the good old MSDOS-days), I was to replace a broken floppy-drive in a computer belonging to some secretary. She had two drives, because some procedures demanded that she was to make backups of floppies at certain stages in her work. So, she would put in a floppy in each drive (she was aware of source- and destination-drives) and then she'd run the copy-command, and so on - no problem, she had been doing that for ages.
>
> Unfortunately, I didn't have a replacement drive that same day, as it would first be arriving the next morning. Neither did I bring a 'blind'-plate to put in place of the empty slot, where the broken drive had been screwed in - and that turned out to be a mistake...
>
> Because I couldn't fix it right away and didn't bother to go back and get a 'blind'-plate, I simply laid out the situation to the secretary and say: 'I'm taking the broken drive out - there is no working second floppy drive, so you *can't* make backup-copies! - I'll be back in the morning with the new drive, alright?' - she said 'sure... '
>
> Returning the next day with the new replacement-drive, she was a bit on her toes. 'Oh, good you're here - I've running a bit behind, in getting the backups back *out* of the computer - can you show me how?... '. Needless to say, I was a bit puzzled.

> It turned out, that she had continued the rest of the day, to put floppies into the empty slot where the secondary drive *had* been mounted! - and then run the copy-procedures. All the 'backup'-floppies where now laying in a tumbled stack down on the motherboard, and now she was concerned getting them out and into her archive-drawer!

What went wrong in the story above, was a mis-conception of the model presented and her not *really* paying attention. The model in her head and the backup-procedure didn't match. The copy-command (MSDOS) would have said something like '*can't find drive B:*', but she must have though: 'It's alright, the apprentice knows where it is! - he said it's on it's way. . . '

Her knowledge about computers and commands to list the contents of floppies apparently wasn't sufficient enough to detect something was wrong. And she was just doing her job as a secretary - and not that of a system-backup administrator. In those days, she wasn't a computer-illiterate at all! - she actually knew more than most secretaries, she just didn't know about *that* particular area.

By taking out the technical wording '*can't find drive B:*' - like Microsoft-products sometimes do, it becomes unclear to specialists and experts what the he. . . ck they're talking about. And it doesn't necessarily present a better and more comprehensible model to the un-knowledgeful user - it's the same model, just without the technical words.

Looking at both the presentation-models of existing solutions and the user-language-issue - it all sums up to this point: Avoid loosing the underlying design-model chosen by the designers - they have experience and have usually thought long and hard about this model. The model is alright - however, the presentations, terms and aids may lack. Additionally, the model is the reality users will have to live within, though the presentation of it can be changed.

So, we suggest a layered approach, which will allow increasing access to details at each layer. A user (or program-plugin) can then choose to operate at whatever layer and level of detail he/she feels comfortable with (or have a need for). Indirectly, some elements of learning and trust comes along with this approach.

When performing higher-level operations, the user can learn about lower-level-details, as shown by the program: E.g. when opening up for a service on some port, the more detailed view will show the exact firewall-rule generated in order to open up the firewall for that access. The possibility to 'see' what the program is doing[1] brings forth some element of trust in the solution. It can be verified by those with insight - and for the vast majority without insight, it will show expertise and action of the program - i.e. '*. . . it does something - and it looks pretty advanced. . .* '.

Finally, a behavioural pattern will emerge over time. This is the result of simply being visual (as discussed in Sec. 2.3.4) and *in-your-face* to the user. Patterns will be recognised by the user as '*the usual way*' of performing, when applying familiar operations. The user will notice abnormalities in the pattern when things starts to go wrong or stops functioning - as the secretary eventually noticed, as her backup-discs didn't come back out again.

If the program is silent and stealthy, no behavioural pattern will emerge - and the human detection-factor will be marginalised. Since no fully automated, bullet-proof, 100%-solution have been invented yet - the most intelligent detection-engine *still* is the human user. The level of 'chatty-ness' (or verbosity) should be adjustable, in order avoid overloading the user with information - but still allow the inquisitive to gain insight (into a desired level of detail). The screen-shot in Fig. 6.2 illustrates the verbosity-idea in the background of a displayed dialog.

6.1 The layered approach

The rationale of our approach, is that the models and views presented to the user must reflect reality. Since several models are in play, operating on different layers of the OSI-model, we will create a layered approach: We will put forward GUI-views with three different levels of detail:

[1]. . . of reading/writing/parsing/generating/informing/warning/alerting/. . .

Basic level: A very detailed and exact view of the raw data in the database. We'll refer to this view as **DBView** (Fig. 6.1).

Structured level: Presentation of configurations, tables, chains, rules and their parameters, in a editable view. We'll refer to this view as **RuleView** (Fig. 6.3).

Elevated level: Condensated/Combined views of rules, processes and the network-layout (as illustrated in Fig 6.5). We'll refer to this view as **ProcView** (Fig. 6.4).

The views presents user-interfaces with increasing abstraction from the nitty-gritty bits, along with decreasing details (and increasing complexity in implementation and effect).

6.1.1 Database view

The DBView (Fig. 6.1) is the most basic level. It contains the SQL-database-functionality and can operate directly on the internal tables of the application. It contains the ER-model of iptables-rules (defined in Sec. 7.4), along with SQL-connection-details (username/password), SQL-queries and -transaction-control.

All table-definitions and static data are defined in various txt-config-files. This is also where the program can load and save the complete data-tables on to disc for transfer to another installed instance. This is also where programmers making new netfilter/iptables-extensions, can add their newly invented parameters, in order to make our parser and generator aware of the new possibilities in the kernel - avoiding the need for re-compiling/-releasing due to extensions.

The transfer of data to and from the database is done using the standard Qt-SQL-module. But all other modules will have to obtain a Qt-SQL-connection-handle from this module using the KParts-interface (explained in Sec. 7.2) in order to use any of the data-transfer-functions from Qt-SQL-module. Otherwise they must know the connection-details themselves (as in Fig. 6.2).

Purpose: The DBView is very raw and low-level, aimed specificly at experts and programmers. It is very detailed and requires knowhow (SQL) and knowledge (netfilter-setup), but it is also very capable, versatile and powerful. Admitted, other more general purpose SQL-clients like *pgaccess* or *mysqlcc* are much more evolved. But they serve a overall SQL all-capability-purpose, and they aren't exactly small embeddable programs. And finally, there was need for some basic SQL-manipulation in our application, so we might as well make it visually available too.

E.g. one could use it with a SQL-query like: "*UPDATE ipt_commandlines_param WHERE (param='–destination' AND value='10.0.0.4') SET value='10.0.0.5'*" to change all occurrences of IP-address 10.0.0.4 to 10.0.0.5 - but <u>only</u> for destination-addresses.

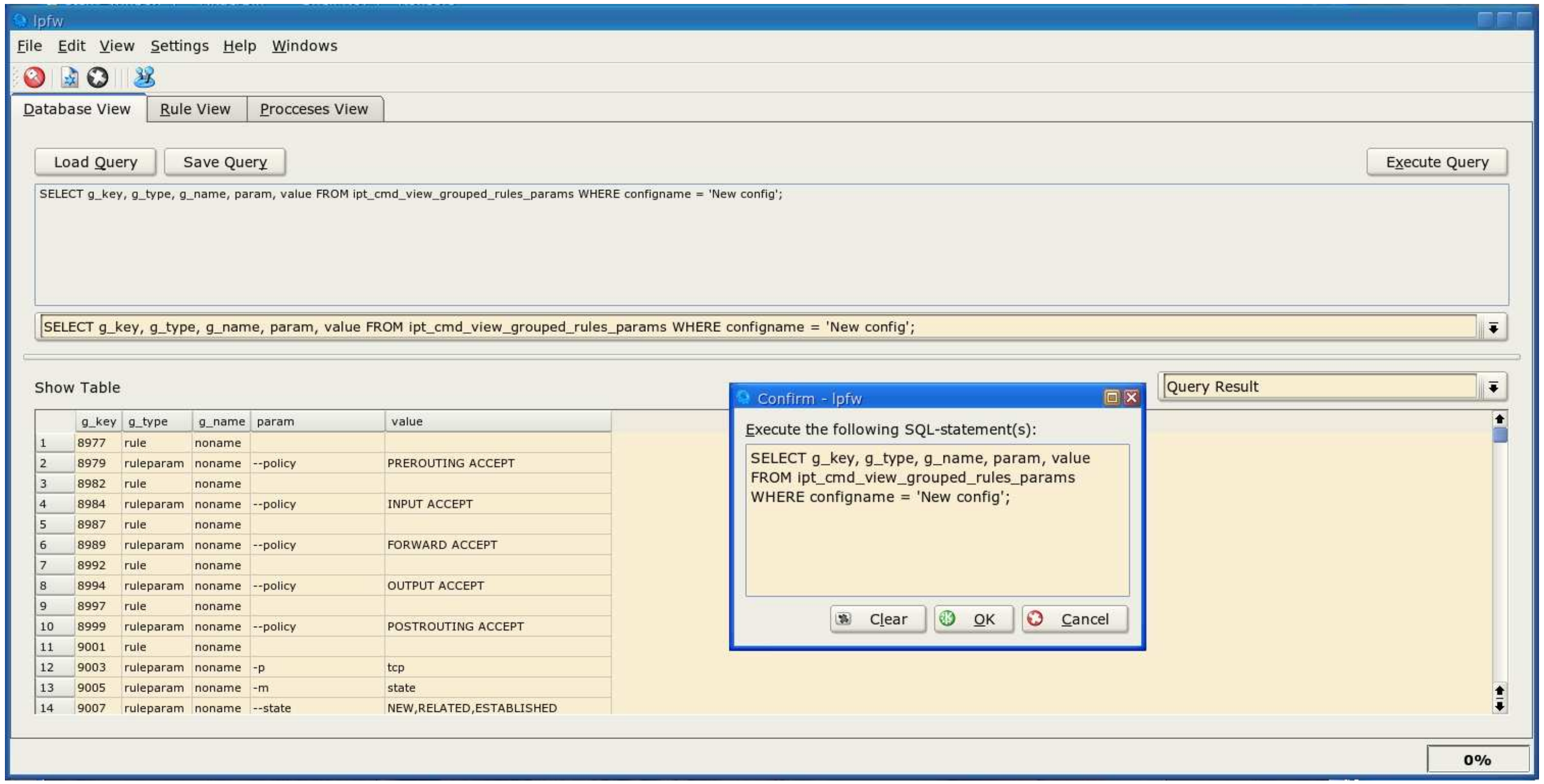
6.1.2 Firewall filter view

A first approach is to show the rules and their organisational structure in a view with rule syntax and context knowledge.

The packet-data traversing the firewall-rules are matched in sequential order to each parameter of each rule. The rules are organised in chain-lists in sequential order, and the chains are order in tables, which again can be said to belong to a particular configuration-setup.

All of this suggests a parent-child-tree relation-ship - as can be provided by a ListView-class from KDE/Qt, we refer to is the RuleView (Fig. 6.3). It's a straight forward presentation as tree-lists are familiar to most users anyway. The treeview presents rules with context-knowledge. It will allow manipulation and edits, but only the possible once - and possibly help where appropriate. This surpasses a standard text-editor-approach, because typos and possibilities can be caught and presented in the context where they occur, when knowledge about state and contents are builtin.

The editing is as expected with context-menus, DragN'Drop, ClickToEdit, etc. Constrains and aid is present when performing operations, where ever relevant and possible (I.e. completion



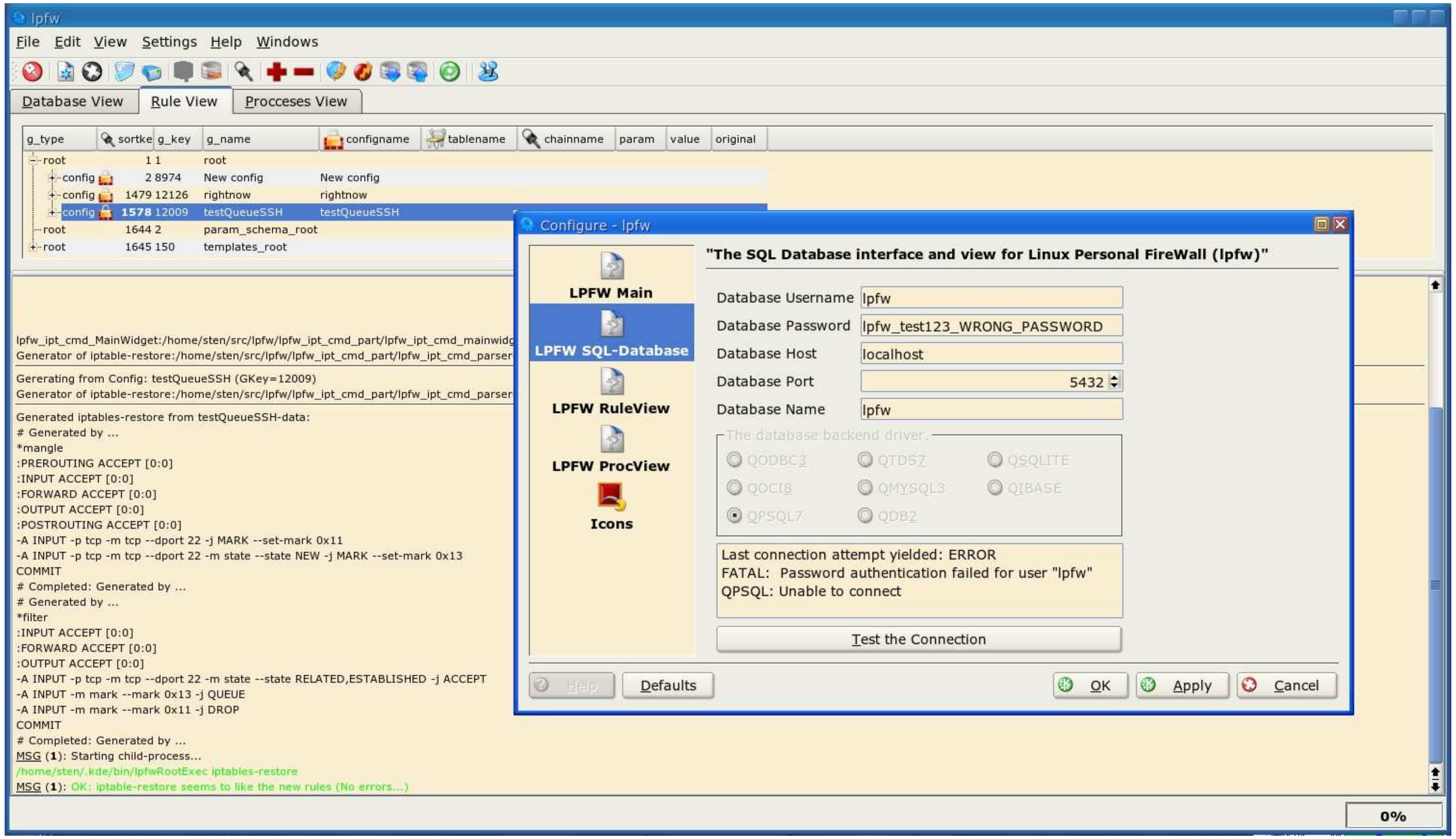
SELECT g_key, g_type, g_name, param, value FROM ipt_cmd_view_grouped_rules_params WHERE configname = 'New config';

Execute the following SQL-statement(s):

```
SELECT g_key, g_type, g_name, param, value
FROM ipt_cmd_view_grouped_rules_params
WHERE configname = 'New config';
```

	g_key	g_type	g_name	param	value
1	8977	rule	noname		
2	8979	ruleparam	noname	--policy	PREROUTING ACCEPT
3	8982	rule	noname		
4	8984	ruleparam	noname	--policy	INPUT ACCEPT
5	8987	rule	noname		
6	8989	ruleparam	noname	--policy	FORWARD ACCEPT
7	8992	rule	noname		
8	8994	ruleparam	noname	--policy	OUTPUT ACCEPT
9	8997	rule	noname		
10	8999	ruleparam	noname	--policy	POSTROUTING ACCEPT
11	9001	rule	noname		
12	9003	ruleparam	noname	-p	tcp
13	9005	ruleparam	noname	-m	state
14	9007	ruleparam	noname	--state	NEW,RELATED,ESTABLISHED

Figure 6.1: DBView (in process of executing an SQL-Query. . .)



g_type	sortke	g_key	g_name	configname	tablename	chainname	param	value	original
root	1	1	root						
config	2	8974	New config	New config					
config	1479	12126	rightnow	rightnow					
config	1578	12009	testQueueSSH	testQueueSSH					
root	1644	2	param_schema_root						
root	1645	150	templates_root						

```
lpfw_ipt_cmd_MainWidget:/home/sten/src/lpfw/lpfw_ipt_cmd_part/lpfw_ipt_cmd_mainwidg
Generator of iptable-restore:/home/sten/src/lpfw/lpfw_ipt_cmd_part/lpfw_ipt_cmd_parser

Gerating from Config: testQueueSSH (GKey=12009)
Generator of iptable-restore:/home/sten/src/lpfw/lpfw_ipt_cmd_part/lpfw_ipt_cmd_parser

Generated iptables-restore from testQueueSSH-data:
# Generated by ...
*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A INPUT -p tcp -m tcp --dport 22 -j MARK --set-mark 0x11
-A INPUT -p tcp -m tcp --dport 22 -m state --state NEW -j MARK --set-mark 0x13
COMMIT
# Completed: Generated by ...
# Generated by ...
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p tcp -m tcp --dport 22 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m mark --mark 0x13 -j QUEUE
-A INPUT -m mark --mark 0x11 -j DROP
COMMIT
# Completed: Generated by ...
MSG (1): Starting child-process...
/home/sten/.kde/bin/lpfwRootExec iptables-restore
MSG (1): OK: iptable-restore seems to like the new rules (No errors...)
```

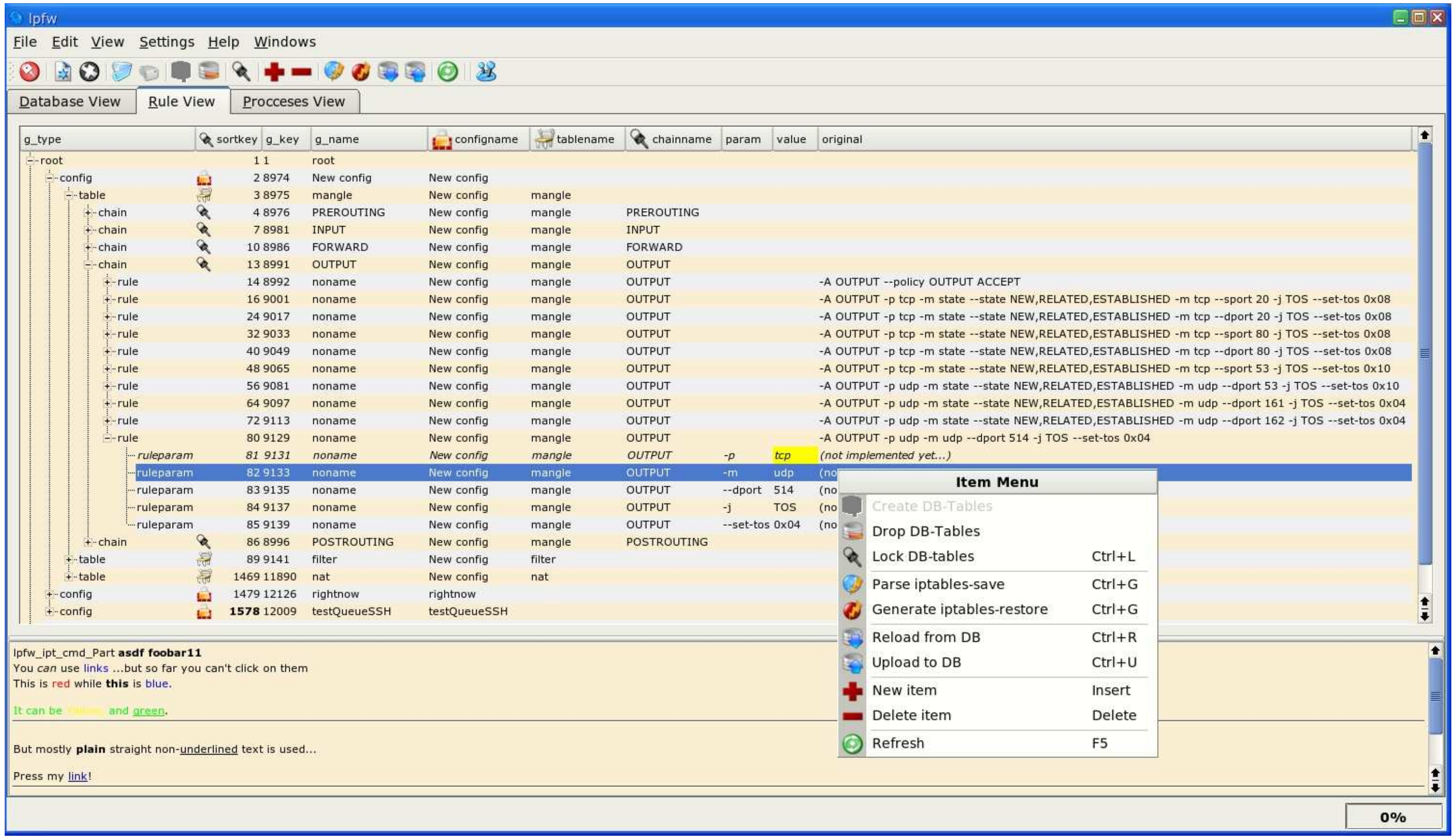
"The SQL Database interface and view for Linux Personal FireWall (lpfw)"

Database Username	lpfw
Database Password	lpfw_test123_WRONG_PASSWORD
Database Host	localhost
Database Port	5432
Database Name	lpfw

The database backend driver:
QODBC3 QTDS7 QSQLITE
QOCI8 QMYSQL3 QIBASE
[✓ QPSQL7] QDB2

```
Last connection attempt yielded: ERROR
FATAL:  Password authentication failed for user "lpfw"
QPSQL: Unable to connect
```

Figure 6.2: Configuration dialog: Connection-details (handles) and verbosity (in the background)



g_type	sortkey	g_key	g_name	configname	tablename	chainname	param	value	original
root	1	1	root						
config	2	8974	New config	New config					
table	3	8975	mangle	New config	mangle				
chain	4	8976	PREROUTING	New config	mangle	PREROUTING			
chain	7	8981	INPUT	New config	mangle	INPUT			
chain	10	8986	FORWARD	New config	mangle	FORWARD			
chain	13	8991	OUTPUT	New config	mangle	OUTPUT			
rule	14	8992	noname	New config	mangle	OUTPUT			-A OUTPUT --policy OUTPUT ACCEPT
rule	16	9001	noname	New config	mangle	OUTPUT			-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 20 -j TOS --set-tos 0x08
rule	24	9017	noname	New config	mangle	OUTPUT			-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j TOS --set-tos 0x08
rule	32	9033	noname	New config	mangle	OUTPUT			-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 80 -j TOS --set-tos 0x08
rule	40	9049	noname	New config	mangle	OUTPUT			-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 80 -j TOS --set-tos 0x08
rule	48	9065	noname	New config	mangle	OUTPUT			-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 53 -j TOS --set-tos 0x10
rule	56	9081	noname	New config	mangle	OUTPUT			-A OUTPUT -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 53 -j TOS --set-tos 0x10
rule	64	9097	noname	New config	mangle	OUTPUT			-A OUTPUT -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 161 -j TOS --set-tos 0x04
rule	72	9113	noname	New config	mangle	OUTPUT			-A OUTPUT -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 162 -j TOS --set-tos 0x04
rule	80	9129	noname	New config	mangle	OUTPUT			-A OUTPUT -p udp -m udp --dport 514 -j TOS --set-tos 0x04
ruleparam	*81*	*9131*	*noname*	*New config*	*mangle*	*OUTPUT*	*-p*	*tcp*	*(not implemented yet...)*
ruleparam	82	9133	noname	New config	mangle	OUTPUT	-m	udp	(no
ruleparam	83	9135	noname	New config	mangle	OUTPUT	--dport	514	(no
ruleparam	84	9137	noname	New config	mangle	OUTPUT	-j	TOS	(no
ruleparam	85	9139	noname	New config	mangle	OUTPUT	--set-tos	0x04	(no
chain	86	8996	POSTROUTING	New config	mangle	POSTROUTING			
table	89	9141	filter	New config	filter				
table	1469	11890	nat	New config	nat				
config	1479	12126	rightnow	rightnow					
config	**1578**	12009	testQueueSSH	testQueueSSH					

Ipfw_ipt_cmd_Part **asdf foobar11**
You *can* use links ...but so far you can't click on them
This is red while **this** is blue.

It can be yellow and green.

But mostly **plain** straight non-underlined text is used...

Press my link!

Figure 6.3: RuleView (in process of changing a parameter in a rule. . .)

of parameters, only drag parameters on to rules, showing changes, etc.). Various colouring and effects are used to highlight changes.

The RuleView is depending on the database-module (DBView), in which the tables have been created. However, the database only creates (and drop) tables, it doesn't have any knowledge of their contents. Once the RuleView have obtained a logged-in connection-handle from the database-module, the RuleView handles transfers of rule-data between the treeview and the database-tables. It will issue the necessary SQL-commands (INSERT, UPDATE and DELETE) when changes are needed, and it can (will) lock the database-tables during synchronising the changes to the database. The lock can also be used as a 1-level undo-function, since the lock is actually the SQL-transaction-control (COMMIT or ROLLBACK) and the user can thereby choose to discard any changes when releasing the lock.

The RuleView will also parse the current firewall-rules in the kernel, and generate the iptables-commands to update the firewall with the new rules. It handles this in a background sub-shell (with root-permissions), by talking to the iptables-commands (IPTABLES-SAVE and IPTABLES-RESTORE). The communication-chat with the commands in the sub-shell can be seen in the log-window below the treeview, both the commands issued and their resulting commandline-output.

The tree can be quite large - with many rules in each configuration, and it can containing several configurations (and possibly belonging to several hosts too). By folding the tree-nodes, focus can be maintained to the area in question. Several RuleView-windows showing different nodes of the tree can be opened at the same time[2] - side by side, just like file-handling using several opened Explore-windows known from MS-Windows. This way, one doesn't have to scroll back and forth, if there is some distance between the drag-source and drop-destination.

Everything that can be performed by the user clicking (or dragging) something - can be scripted. The RuleView implements all functions[3] from the user-interface as DCOP-functions too. E.g. to remove a line, the user would press the delete-key (or icon), but it can also be done by this bash-command: 'dcop lpfw LPFWRuleView deleteNode <*nodeId*>'[4]. Same goes for inserting, editing, searching nodes - or for committing to database, push the new rules into the firewall, etc.

The script-able DCOP-interface serves several purposes.

1. It allows automation and repeatable playback of actions, and such is always appreciated by administrators and advanced users et al. As a spinoff-effect, it also enable automated regression-tests to use and exercise the functions in the view.

2. It opens up a interface for other modules to use, which *isn't* compile-time dependent. That allows other views (E.g the ProcView) to insert/change rules and push the new rules to the firewall, without being linked to the RuleView-component.

3. It allows improvement of interfaces and APIs: I.e. by havning extra service-functions, a higher level of abstraction is present in our RuleView, than talking to directly to the IPTABLES-command. E.g. it becomes possible to search in the current rules using regular expressions (for instance), something which isn't possible with the iptables-command.

4. By being reachable from e.g. bash or C++, the DCOP-interface makes our program the only 'official' C/C++ - (or scriptable-) interface to iptables/netfilter[5]!

Purpose: The RuleView's overall purpose becomes this: It handles the rules in the database and the kernel - it brings context and syntax of iptables-rules into the picture.

I.e. it stores and manipulates configurations of rules, parses and pushes configurations to the kernel, shows changes of which parameters, on which rules, on which configurations, in what sequence etc.

[2]...although it's buggy right now, only 'drag-copy' works - 'drag-move' doesn't.

[3]...at least, it will do so very soon - right now *most* of the actions are reachable by DCOP-scripting.

[4]See the man-page for more on the dcop-command, or use the GUI-version: kdcop
- or simply try this: 'dcop kdesktop KBackgroundIface changeWallpaper'.

[5]according to netfilter FAQ - issue 4.5 "Is there an C/C++ API for adding/removing rules?" (http://www.netfilter.org/documentation/FAQ/netfilter-faq.html)

The RuleView requires less expertise than the DBView (which isn't friendly at all), but still some general knowledge of TCP/IP-networks and netfilter-rules is required. The presentation abstracts away the SQL-background and aids in keeping rule-structures intact, by listing the configuration, sequences and organisation of rules. It present possible options for changing the firewall (to local- or remote-hosts), and it creates a place to build and share template-sections of rules (e.g. anti-spoofing-rules, no-broadcast-rules, no-TCP-SYN-flood-rules (DDoS-attack), etc.).

Finally, later it will be used as a giant context dialog for other views, when they perform changes. Because, the RuleView can show the context of changes and roll them back if the user dosn't want them.

6.1.3 Process communication view

The final piece in the puzzle is getting the firewall to respond to packets for which it currently doesn't have particular rules for. This will make the nature of the firewall appear more dynamic. When certain packets are matched by newly developed rules, a popup-dialog should ask the user: 1) if the connection should be allowed or disallowed?, 2) only once? or always from now on?, 3) for this program only ("Netscape-traffic")?, or by the nature of this program ("port 80-traffic")?, or by this host, subnet or...? - the possibilities are limited only by what we can get iptables to generate rules for.

From the user's point of view, the purpose isn't about the rules as such - they are just means to the goal. The purpose is to control the programs (processes) performing communications. So, a link between the rules, the packets of a communication-session and the program-process involved is needed, and that is what the ProcView (Fig 6.4) does.

The link can be achieved by layering on the firewall-filtering-model to the process view presented by e.g. PS/LSOF and it's GUI-front-ends from KDE/GNOME[6]. It brings a familiar tree-view of what programs are currently running on the host - and in our case: only programs doing network-communication.

The ProcView can show processes along with their communication-data (sockets). The rules which opens up the firewall for a particular process is[7] presented as sub-nodes right below the particular process - as illustrated at the bottom process in Fig 6.4.

Initially, not all processes may have rules to govern them, as e.g. is the case of newly installed applications. To resolve this problem, we capture all packets in the firewall (See how in Sec. 7.7 on page 80), after all other rules have been checked and right before they get default-dropped. At that point, all checks have been made to see, if the packet belongs to a processes for which we *have* rules. At this point, we have a packet which is un-dealt with by any rule - possibly for a new program perhaps?

These captured packets are queued into a single connection-attempt[8] - as shown in Fig 6.4. Each connection-attempt is halted by the firewall, and listed in the view until the user clicks on the ACCEPT- or DROP-icons. The connection-attempt states which process will get the packet - if allowed though. It also state where it's coming from, all though that is more unreliable due to ip-spoofing etc. But, if a user expects a particular connection e.g. an ftp-connection from some known source, it is possible to let it though, without making a permanent rule for ftp-connections - it's just this one time[9].

[6]and TaskManager on Windows.

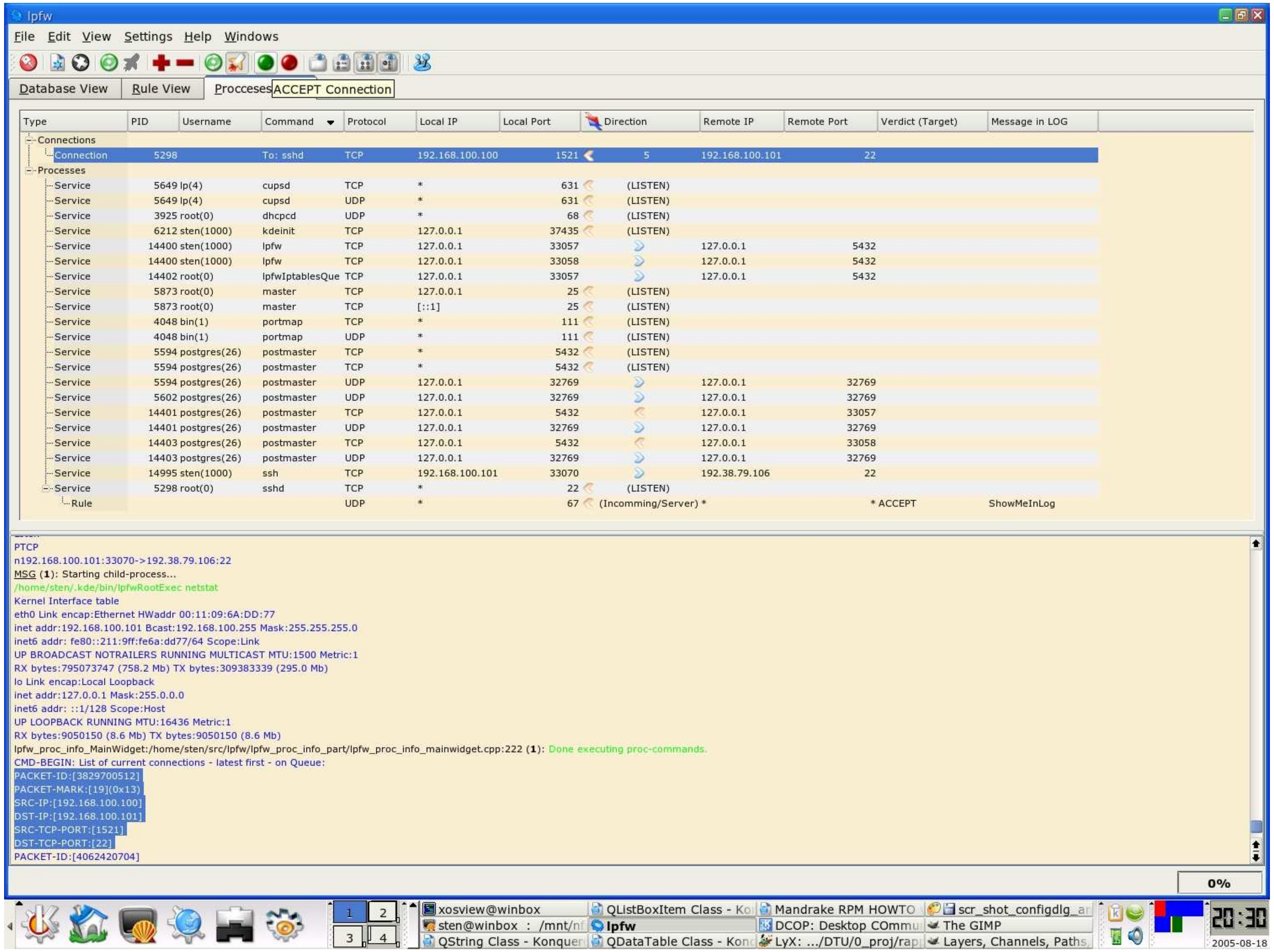
[7]Please read: will be very soon - when the SetupWizard is done...

[8]Several packets may be sent to the same process, i.e. SSH will repeatedly send new packet when the first one times out (due to no response).

All these packets are from the same channel (See also Sec. on page 15), because they have the same socket-data (src/dest -ip and -port).

The number of packets can be seen in the 'Direction's column, the socket-data makes up most of the other columns.

[9]That is due to netfilter being a stateful-packet-filter: All further packets to an initial connection-packet, can be allowed though with a match-rule of '–state ESTABLISHED, RELATED –target ACCEPT'. An initial connection-packet (such packet has the TCP's SYN-flag set), can be matched with e.g. '–state NEW'.



Ipfw

Database View | Rule View | Procceses | ACCEPT Connection

Type	PID	Username	Command	Protocol	Local IP	Local Port	Direction	Remote IP	Remote Port	Verdict (Target)	Message in LOG
Connections											
Connection	5298		To: sshd	TCP	192.168.100.100	1521	◀ 5	192.168.100.101	22		
Processes											
Service	5649	lp(4)	cupsd	TCP	*	631	◀ (LISTEN)				
Service	5649	lp(4)	cupsd	UDP	*	631	◀ (LISTEN)				
Service	3925	root(0)	dhcpcd	UDP	*	68	◀ (LISTEN)				
Service	6212	sten(1000)	kdeinit	TCP	127.0.0.1	37435	◀ (LISTEN)				
Service	14400	sten(1000)	Ipfw	TCP	127.0.0.1	33057	▶	127.0.0.1	5432		
Service	14400	sten(1000)	Ipfw	TCP	127.0.0.1	33058	▶	127.0.0.1	5432		
Service	14402	root(0)	IpfwIptablesQue	TCP	127.0.0.1	33057	▶	127.0.0.1	5432		
Service	5873	root(0)	master	TCP	127.0.0.1	25	◀ (LISTEN)				
Service	5873	root(0)	master	TCP	[::1]	25	◀ (LISTEN)				
Service	4048	bin(1)	portmap	TCP	*	111	◀ (LISTEN)				
Service	4048	bin(1)	portmap	UDP	*	111	◀ (LISTEN)				
Service	5594	postgres(26)	postmaster	TCP	*	5432	◀ (LISTEN)				
Service	5594	postgres(26)	postmaster	TCP	*	5432	◀ (LISTEN)				
Service	5594	postgres(26)	postmaster	UDP	127.0.0.1	32769	▶	127.0.0.1	32769		
Service	5602	postgres(26)	postmaster	UDP	127.0.0.1	32769	▶	127.0.0.1	32769		
Service	14401	postgres(26)	postmaster	TCP	127.0.0.1	5432	◀	127.0.0.1	33057		
Service	14401	postgres(26)	postmaster	UDP	127.0.0.1	32769	▶	127.0.0.1	32769		
Service	14403	postgres(26)	postmaster	TCP	127.0.0.1	5432	◀	127.0.0.1	33058		
Service	14403	postgres(26)	postmaster	UDP	127.0.0.1	32769	▶	127.0.0.1	32769		
Service	14995	sten(1000)	ssh	TCP	192.168.100.101	33070	▶	192.38.79.106	22		
Service	5298	root(0)	sshd	TCP	*	22	◀ (LISTEN)				
Rule				UDP	*	67	◀ (Incomming/Server) *			* ACCEPT	ShowMeInLog

```
PTCP
n192.168.100.101:33070->192.38.79.106:22
MSG (1): Starting child-process...
/home/sten/.kde/bin/IpfwRootExec netstat
Kernel Interface table
eth0 Link encap:Ethernet HWaddr 00:11:09:6A:DD:77
inet addr:192.168.100.101 Bcast:192.168.100.255 Mask:255.255.255.0
inet6 addr: fe80::211:9ff:fe6a:dd77/64 Scope:Link
UP BROADCAST NOTRAILERS RUNNING MULTICAST MTU:1500 Metric:1
RX bytes:795073747 (758.2 Mb) TX bytes:309383339 (295.0 Mb)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX bytes:9050150 (8.6 Mb) TX bytes:9050150 (8.6 Mb)
Ipfw_proc_info_MainWidget:/home/sten/src/Ipfw/Ipfw_proc_info_part/Ipfw_proc_info_mainwidget.cpp:222 (1): Done executing proc-commands.
CMD-BEGIN: List of current connections - latest first - on Queue:
PACKET-ID:[3829700512]
PACKET-MARK:[19](0x13)
SRC-IP:[192.168.100.100]
DST-IP:[192.168.100.101]
SRC-TCP-PORT:[1521]
DST-TCP-PORT:[22]
PACKET-ID:[4062420704]
```

Figure 6.4: ProcView (in process of accepting an incoming SSH-connection. . .)

In the log-view below the connection- and process-trees, the usual chatty-ness can be seen: output of PS, LSOF, NETSTAT and the packet-capture-program (the queue-handler) can be seen.

The DCOP-interface is little evolved, but it could provide a user-space script-able interface to access the packets and analyse the contents, since we have captured both TCP/IP-headers and application-payloads of the packets. Or it could keep track of running processes, and when they terminate: Close the related ports in the firewall (E.g. for user-space NFS-servers, P2P-programs etc.).

Purpose: The ProcView is an more friendly view to the average user (and friendly to programmers too;-). It shows processes and connection-attempts in context with the firewall-rules[10]: Which program is requesting communication and should it be allowed? (ACCEPT-/DROP-'ed). And should a permanent rule be made - handling all future instances of this type of communication? (inject/edit sections in the RuleView though DCOP).

The ProcView uses the RuleView's DCOP-interface to find, analyse, alter and inject rules (or if necessity dictates: the database-tables directly).

6.1.4 Total network model

An more elaborate approach must envision the combined models of A) firewall netfilter, B) the OSI-layered network stack and C) the network-layout and routing detected on the firewall host. It envisions the complete path travelled by a packet thought the system, from arrival on a network interface to (possible) delivery to a process. The picture to depict in the user's mind is the following figure (Fig 6.5):

[10]...please read: will soon - when the SetupWizard is done.

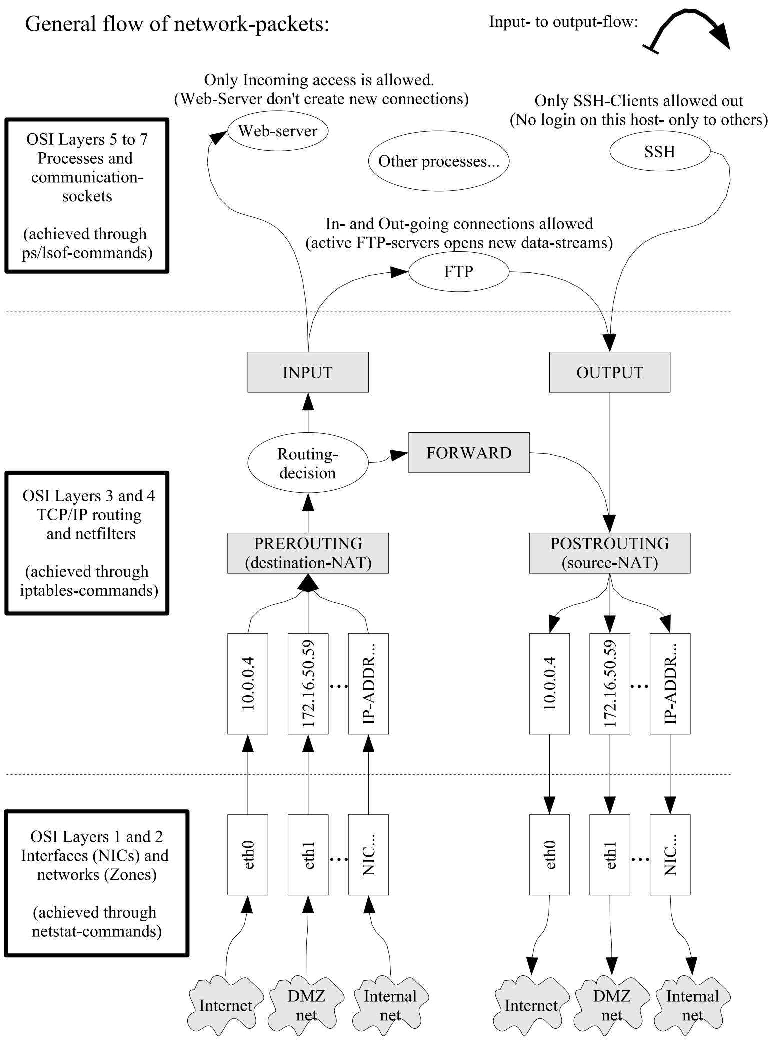
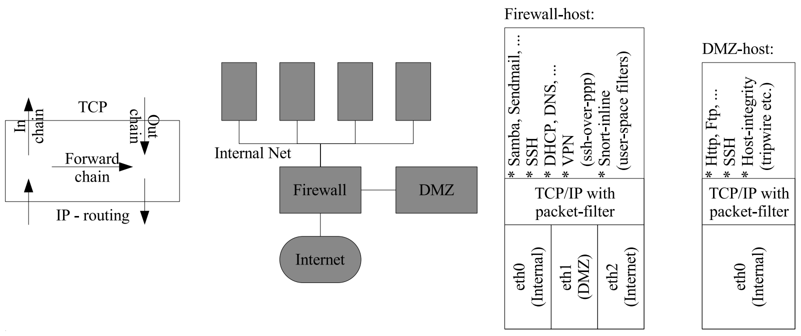

General flow of network-packets:
Input- to output-flow:
Only Incoming access is allowed.
(Web-Server don't create new connections)
Web-server
Other processes...
Only SSH-Clients allowed out
(No login on this host- only to others)
SSH
OSI Layers 5 to 7
Processes and communication-sockets
(achieved through ps/lsof-commands)
In- and Out-going connections allowed
(active FTP-servers opens new data-streams)
FTP
INPUT
OUTPUT
Routing-decision
FORWARD
OSI Layers 3 and 4
TCP/IP routing and netfilters
(achieved through iptables-commands)
PREROUTING
(destination-NAT)
POSTROUTING
(source-NAT)
10.0.0.4
172.16.50.59
IP-ADDR...
OSI Layers 1 and 2
Interfaces (NICs) and networks (Zones)
(achieved through netstat-commands)
eth0
eth1
NIC...
Internet
DMZ net
Internal net

Figure 6.5: Complete firewall-network-model of a host (illustration).

The illustration of Fig 6.5 are showing the key-components needed from the three sections of OSI-layers: the datalink-layer (1-2), the network-layers (3-4) and the application-session-layers (5-7). This will be reflected later in the system design overview (Fig 6.9), where the input- and output-flows to and from the system, uses these hooks into the three OSI-sections of the network stack.

The composite model can be seen as a merger of three separate views, each coming from the sections own way of modelling their internal structures (Fig 6.6):



(a) 1) netfilter-model, 2) physically wired-view, 3) and 4) OSI-7-layered view of a host

Figure 6.6: Three models of networking on a host (prototype-sketch).

The three models (Fig 6.6) are showing: Left) netfilter's own internal model of it's IP-packet-filter, Middle) the network-zones, and Right) how a host in a zone have an interface connected to the network-zone, along with some communication-processes running on that host.

Graphical View (not implemented) **Note:** The graphical graph-view never got implemented - due to too many unresolved questions and too little time (isn't such always the case?). Some reasons are discussed below and additionally in Sec of 8.1 on page 84.

The more combined view could be realised by a graph using the dot-tool[11]. In such view, a user might look at the firewall-setup like in Fig 6.7:

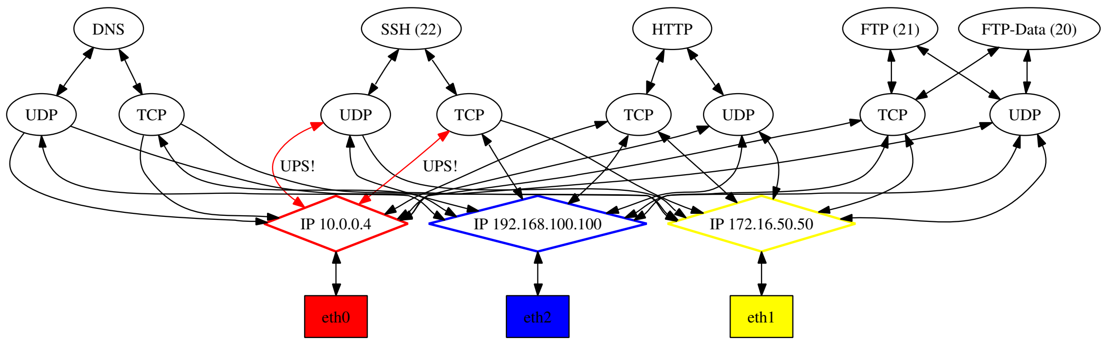


Figure 6.7: Simple process view (prototype-sketch with dot).

While this simpler view offers good understanding processes to NICs, it doesn't distinguish between local- and network-wide- processes. That is determined by routing, which isn't in play in Fig 6.7.

Adding the netfilter-model indirectly shows more information on routing too, since it is implied by the routing decision and NAT'ing in netfilter. Combined they might look like in Fig 6.8:

[11]From the Graphviz package (http://www.graphviz.org/)

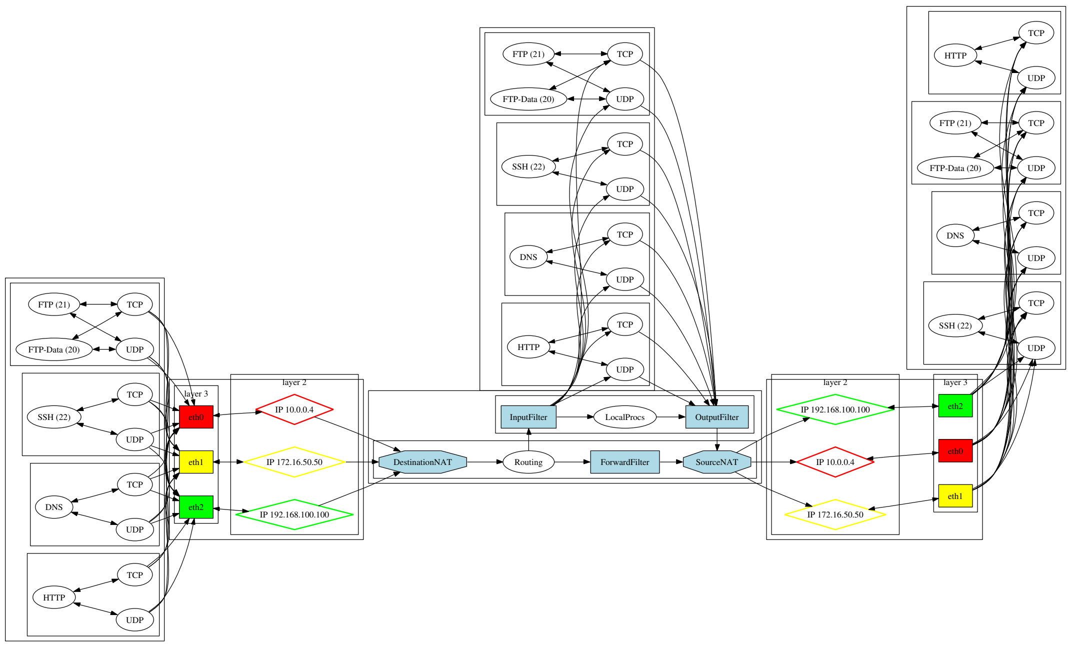


Figure 6.8: Advanced process view (prototype-sketch with dot).

The advanced view offers the link between the netfilter-model, network-stack and the process-view. How specific the graph should be depends on the size of it. It should be able to fit into the screen in order for the user to maintain an overview of things. The rule of a netfilter-firewall can easily reach 700+ lines, not easily comprehended or altered. Therefor the GUI must compress the information into much less space, by use of graphs, sub-information concealment and explosion and by clever presentation of only the interesting information - what ever that is.

To make it fit, more or less elaborate information can be displayed. By collapsing some of the information, a simpler and more presentable graph can be generated. E.g. maybe it isn't interesting to see the links between network-devices and -interfaces (*ethX <-> ip_X*) or between a service (*SSH*) and the TCP-protocols (*TCP, UDP, RTP, IGMP* etc.). For the user to get some more details, clicking can be used to explode into sub-graphs again (*ethX <-> ip_X*) or by popping up dialogs showing the details. The user could also choose to minimise the view of specific details and sub-graphs by making them as 'seen' or 'hide', thereby inspecting details and once seen - they can be collapsed. Upon changes in collapsed parts they could be exploded back again.

Flexibility is also an issue. The netfilter software is quite extensible and new extra modules from third parties occur regularly. The GUI must equally extend this option, by allowing GUI-code to be compiled and distributed independently of the netfilter module, so that no re-compiles are necessary. E.g. The GUI could use some form of XML to build the graphs, because that would require simple text-file-patching when updating.

A Futuristic User Session (Use-case) Putting the above together, the user should be able to operate the solution by dragging and cutting lines on a condensed view-graph presenting the compressed information. Lines on the graph represent rules, and nodes represent entities like netfilter-chains, NICs and processes. Right-clicking should provide access to intricate details (information-explosion) in e.g. an new sub-view or in a dialog (dynamically generated) - thereby giving the flexibility and extendibility. The GUI shouldn't hide what is being generated behind the scenes, it should have layers of API-interfaces and stages of execution where external programming hooks can be attached.

6.2 System Design (Overall Design)

Our system design must support our desired user-interface-handling (as presented in the previous of Sec. 6). And it must also conform to our technical needs of inter-operability, build-independency, and the general layered approach of modularity through open API's .

In essence, the purpose of the application is to make the link between:

A) Packets in the TCP/IP-stack,

B) The rules in the firewall filtering the packets and

C) The processes performing the communications - i.e. 'to whom the packets belongs to, and the rules they are getting governed by'.

The above informations are tapped from (and injected into) the system by hooks at the network-stack (OSI-layers 2-4), the process table (OSI-layers 5-7) and the netfilter firewall. This involves all three sections on illustration in Fig. 6.5 on page 57. The key operating-scheme of the program is like this:

1. Query the iptables-firewall for currently installed rules.
2. Query the iptables-firewall for packets captured and on hold for a decision.
3. Query the host for running processes that performs communication and obtain their socket-data.
4. Feed the iptables-rules and the process-information to a database.
5. Present the database-relations in views: Show the current rules and allow/perform manipulation of the rules. Show the relation the process-information to these rules. And relate the process-information the captured packets - relating their origin/destination, purpose and context to the programs performing communications.
6. Generate new rules from the manipulated database-relations, and execute decisions on captured packets - thus allowing the programs to communicate only as specified.

Complete the above steps as often as desired - allowing maintenance (roundtrip-engineering) of the firewall.

When manipulating the rules, the presentation-views, wizards and verifiers are present to aid the user. All of these will be module-parts and plugin-extensions, so that they can be developed, modified (or substituted) and released independently of the main application framework.

Additionally, the personal touch of popping up connection-attempts is the issue of extending the firewall from (statefull) packet-filter to circuit-level (see Sec. 2.3.4 on page 20). It draws on the view processes (with their sockets) to show the connection-attempt in that context: "To whom, from whom". It relates to rule-manipulation, by being a rule itself (the netfilter/iptables QUEUE-target-module in standard releases), and it may insert rules to allow/deny the connection-attempt permanently.

At this stage, the main focus points have been touched. It is now possible to compose a overview of the necessary functionality - grouped into modules.

An overview is provided in Fig 6.9:

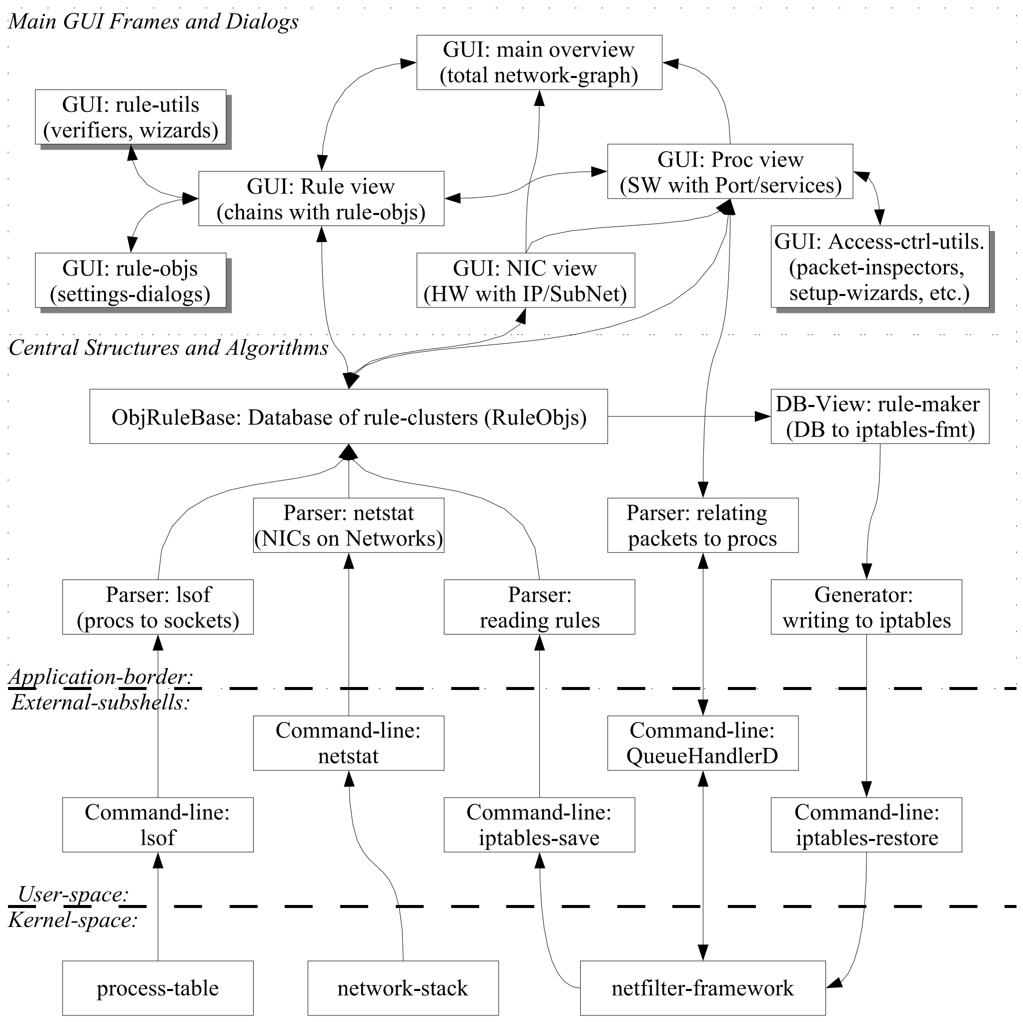

Main GUI Frames and Dialogs
GUI: main overview (total network-graph)
GUI: rule-utils (verifiers, wizards)
GUI: Rule view (chains with rule-objs)
GUI: Proc view (SW with Port/services)
GUI: rule-objs (settings-dialogs)
GUI: NIC view (HW with IP/SubNet)
GUI: Access-ctrl-utils. (packet-inspectors, setup-wizards, etc.)
Central Structures and Algorithms
ObjRuleBase: Database of rule-clusters (RuleObjs)
DB-View: rule-maker (DB to iptables-fmt)
Parser: netstat (NICs on Networks)
Parser: relating packets to procs
Parser: lsof (procs to sockets)
Parser: reading rules
Generator: writing to iptables
Application-border:
External-subshells:
Command-line: netstat
Command-line: QueueHandlerD
Command-line: lsof
Command-line: iptables-save
Command-line: iptables-restore
User-space:
Kernel-space:
process-table
network-stack
netfilter-framework

Figure 6.9: System overview..
(See also Fig. 6.5 on page 57 for relations to the OS(I)-network-stack.)

The design relies on certain key properties, which can be summed up below:

- It is for ordinary users on Linux-platforms, so any tools, libraries and components used, must be readily available in any decent (and recent) Linux-distribution (RedHat, SuSE, debian, ...). No special re-compiled patches should be needed.

- The use of a Database is central in the design.
  Firstly, it allows much larger amounts of data to stored (i.e. it scales better) and it allows cross-searching much more flexible that linked lists of objects (i.e. an STL-list containing strings of iptables-command-lines). This is what databases and SQL is designed for in the first place.
  Secondly, by using a standard SQL-base, an established interface is used - no re-invention of the wheel here. Whatever and however the GUI operates on data, it has no impact on how parser- and generator-tools populate and traverses the database. Likewise, GUI-extensions and algorithms (Verifiers, Dialogs, Wizards, Views etc.) can use SQL-queries to extract the essential data only - or operate on all of it.
  The result is that the database makes the parser- and generator-tools towards the system independent of the GUI-tools. And any GUI-extension have access to all the data they need,

since it isn't possible to predict what data and interface should be made available to an future GUI-extension. And all in a standardised way: SQL.

- Dynamic XML-dialogs: This feature should allows us to create and distribute binaries of this project without the need to recompile individual modules handling a particular netfilter-module. The netfilter-modules are either part of the netfilter-package and is distributed a vendor, or the modules are created by a third part. It isn't likely that they would know about the GUI-wrappers in this project. Therefore, this feature is paramount and necessitates the use of KDE/Qt, which have built-in support for such feature.

- Display and interaction of nice and presentable graphs: The need for a simple and pleasant view of the packet-flow is important. This is a direct measure of overview and usability which is one of the goals of this project. A firewall design tool could be seen as a editor of flow-diagrams for network-packets, and flow-diagrams are graphically represented using directed graphs.
  The *dot*-tool in the Graphviz-package contains several tools for working with directed graphs with long history and proven track-record stability-wise.
  The *tulip*-package is a larger and more extensible package that might be used instead since it seems to have pre-fabricated GUI-viewers and editors, along with the possibility of importing dot-graphs.

- Parsing (transformation) of data between the GUI-views, the database and system-commands (e.g. iptables): This means that we must understand (parse) the output of iptables to create (generate) the condensed graph to the user. The reverse step includes parsing the graph and create the necessary iptables-commands. The choice of parser-tool could have been *Lex/Yacc* (*flex/bison*). Or the more modern *ANTLR*, since it generates both Java- and C++-code with good debugging facilities using Java (with display of the parser's internal abstract symbol-tree, AST), while retaining the C++-capability.
  But currently, the parsing have been done 'by-hand', using regular-expressions (QRegExp from Qt), since the parsed data haven't been complex in states (the yacc-part), only in amount and symbol-diversity (the lex-part). And QRegExp is a fast and flexible class with capabilities close to those of e.g. sed/awk.

Chapter 7

Detailed Design issues

Here follows more detailed discussions on the design-choices and technologies deployed.

7.1 Knowledge discovery and acquisition

Some things may not be rocket science, but for the un-enlightened everything *is* difficult - i.e. "*The devil is in the details...*".

This incomprehensive list of issues are illustrating that point, in practical projects like this one:

- The development is taking place on Linux, using Open- and Free-Source. That added an extra dimension as far as documentation and "knowledge finding", because the API-, tutorial- and background-documentation of kernels and libraries are ranging from excellent to non-existing. Some documentation is outdated - the source have moved on, some just not there - read the source-code.
  That is different from commercial products, which strives to have uniform and updated documentation - although the actual descriptions may not be any better, at least it is there. One does not seek or read in vain, is more simple: "if it ain't right here - we haven't documented it at all."
  (Illustrated by the need to read multiple KDE-tutorials about the same issue, and read it along with the source-code - and then cross-correlate the information to filter out errors and uncover the issue.)

- As for Components and third party libraries (APIs): Knowing your components beforehand is an good idea. Otherwise one tends to go with the first component or API-library found for the task. But choosing the right components are essential. Remaking that choice may be initially wasteful, but later on it will have a heavy impact. Remember that the better API will be used all the time during development, and the inferior choice will have an exponential impact on effort and time - bad components and APIs may explode in difficulties, while good ones allows time and effort to be focused elsewhere.
  Conclusion is: spend a little more time initially on getting the component-choices right - it will come back many-fold in saved time and efforts, increased functionality, better overview and planning-estimates - simply less problems and frustrations.
  (Illustrated by the selection-process of a good database-API. Time was spent on trying out several libraries, and Qt turned out to have the better one - and it have, despite some inconsitentciesinconsistencies, saved <u>many</u> working hours later on.)

- Some components are white-boxed - others are black-boxed. Examples are the KDE- and Qt-libraries. Meaning, the documentation on some components (like KDE) are not sufficient, and one needs to read the code of the component - which makes it a white-box. Others (like

Qt) have excellent tutorials, examples and reference documentation and can be used without a need to know whats in them - i.e. black-boxed.
Using white-box components are only slightly better that ordinary code of any kind. It is only the design-part, which may have turned the component into a self-enclosed code-entity, where ordinary code may not have been crafted with such in mind.
As always - ones millage may vary - and the conclusion is that: without good documentation, components are only half a step up the software-development and -maturity ladder. Using them give you tested capability right out of the box, but the time invested in finding out how to use them, are same as with ordinary code - i.e. you got to deciferdecipher the design though reading the code.
(Illustrated by the usage of KListView-classes as the main view-widget in most of the components. It has very scares documentation and 5000 lines of code. In order to be able to use it, one must read and use the Qt-counterpart QListView first, in order to understand the new set of functions on the derived KListView - how to use them and why they are much better.)

The above points are present to some degree in every project, but was profound through out this project. It resulted in more time used on the above issues than initially expected - and that is why *the devil is in the details...*

The remaining chapter deals with specific details encountered during the implemnetation-phase. The topics mostly follow the modules-bounderies of the system overview in Fig 6.9. We start off with the technology which enables modules in the first place (along with 3.-party extensions), - one of the core requirements of this project.

7.2 Plugin-system

It is important for the system, that plugins must be capable of loading and running code complied independently of this application-build, and that the application-build is independent of the kernel-build. This constitutes the issue of ”the dynamic-dialog-aspect” from the requirements and it is indeed one of the core requirements. The reason for this importance is threefold (we recap from the requirements):

1. The kernel: Rebuilding is not feasible just because the kernel have been recompiled with an extra extension. It must be possible to change the kernel from one version to another, without recompilation (or loss of functionality) of end-applications. Just because, iptables (or netfilter within the kernel) changes and get extended, our application must not need a re-compile.

2. The end-users: Users don’t like to re-build the application when ever the kernel change. For the vast majority, this just will not happen - since they don’t know how to compile and install a source-package.

3. The 3. party contributers: The *can* re-compile as crazy as they like - and does knows how to... but they cannot deploy their compiled code. Everybody else would have to re-install the whole package (of the main- and all 3. party-components), and it would have to be re-released from one single monolith entity - the one who made the compile!

Since a plugin may be build independently, the main application does not know about any class-definitions or header-files outside it self, and it must link to an shared library at runtime. This poses two overall problems: 1) how to find the library, 2) how to load and link in the object code in it.

7.2.1 The linkage-problem

The problem of finding, loading and linking the library is a task that can be solved by **A)** the main application (a custom solution), **B)** the OS (the library loader - 'ld' on Linux) or **C)** an existing API capable of such (the KDE-libs). As outlined in our strategy, we will try and avoid making it our self (solution A), and also: this problem is a very difficult topic requiring lots of specific OS- and compiler-expertise - which this author simply don't have enough of. So that basically leaves us with solution B or C.

The C++-standard does not define how the object-parts of functions and classes should be marked or labelled inside the library. This is referred to as *name-mangling.* Therefore, it isn't possible to compile C++-code into a library with one compiler and link to that library with a different compiler, because they may use different function-labels in the compiled object-code for the same C-source function-declaration.

E.g. some function like "*MyClass::foobar(int arg)*" may be marked as "*__MyClass_::_foobar(++int SYMBOL_XYZ)*" by a Borland-compiler and as "*++MyClass_::_foobar(<<large>> SYMBOL_ABC)*" by a Microsoft-compiler.

But, because the C-standard *does* define name-mangling, the OS-library-loader can link standard straight C-code (marked as '*extern "C"* {}') - but only straight C. Since we will be using KDE/Qt, lots of object-oriented C++-code are present in our libraries, and therefore solution B (of using straight standard C) - is too low-level for our needs, so we have to go for solution C (the KDE-solution).

In essence: We are force to use string-based casting between classes, because we make straight C-calls to libraries. Standard C doesn't know about C++-classes, so everything is casted to some address-pointer (void*). We use C, because the OS-linker-loader can load and link standard C-labelling - but it cannot load and link a arbitrary shared library with C++-labelling, because the labelling of classes and functions inside the library may/does not correspond with the labelling given by the caller (different compiler/version).

So, finding a function's entry-point is not possible for the OS-linker - not with different compilers installed, and not in a portable way for different platforms. The source-code may be the same given to compilers, but the resulting object-code-labelling inside the libraries will not be the same. And 'newly' compiled programs may use a differently labelling-scheme and request these labels from the OS-loader-linker - which cannot find the labels in the 'older' differently labelled library-object-code (previously produced by another compiler/version).

The result is, that *only* straight C-calls can be made between libraries, *if* the libraries are to shared libraries (.so-libraries), dynamically linked - and not staticly linked. And we *need* the dynamically linked aspect for our plugins and extensions.

7.2.2 The KDE solution

The action of getting KDE to handle our linkage-needs, are generally referred to by KDE as making *modules* or *parts* (modules, KParts, KPlugins, KCM's, KDocklets... - 'many names for a beloved child'...). It allows us to register our libraries within KDE (so we can find them), and getting KDE to load and link the objects into our application at runtime. The KDE system actually uses the OS-loader-linker ('ld') underneath, but to the programmer it provides a system so that forced compiler-casting are done as safely as possible, since the compiler cannot be used to check it - because of the mangling differences.

This is all done by an elaborate system of correct naming of factory-functions ('void *init_*name*()', marked as '*extern "C"* {}') with in shared libraries ('lib*name*.so', from the Makefile.am), '.desktop'-ini-files (for registration of *name*) and resource-flies ('*name*rc'-config- and '*name*.rc'-xml-files, for GUI-layout etc.). Exactly how this is done, is a very lengthy discussion, much better explained by the KDE documentation, which contain user-guides and reference-documentation needed to understand the details.

The KDE-solution gives us many parts and abilities:

- KParts (modules containing GUI-widgets) and KPlugins (modules without any GUI-widgets),
- KConfigModules (aka KCM - KParts with automatic plugins for config-file/-dialog handling across part-boundaries),
- KXMLGuiBuilder, KPartManagers, etc. handling the dynamic merging of widgets, menus, toolbars from the parts into the application (when the part has focus the menus are merged in, and when it looses focus. . . etc.).
- KConfig XT - a XML-Gui-system composed of kconfig_compiler, KConfigDialog and KConfigSkeleton, which enables simple XML-specification of all config setting, everything from dialog to file handling is done by the kconfig_compiler and the classes.

KDE-System-Configuration-Cache (KSyCoCa) All parts are registered through their '.desktop'-file which is read into the KDE-System-Configuration-Cache (aka KSyCoCa) by the 'kbuildsycoca'-command. The KSyCoCa maintains an LDAP-like, binary-, read-only-, quick-lookup-database. The KSyCoCa can queried by parts and applications for abilities, and requested to link the libraries and load the class-factories within the libraries, in order to get the object-code of a part with that particular ability.

This is possible because '.desktop'-files hold informations like the part-name (e.g. '*mypart*'), the library-name (e.g. 'lib*mypart*.so'), the class-factory within (e.g. '*MyPart*'), the data it can handle (e.g. 'mime/mp3', '.mp3'-extension), it's possibly parent-application (e.g. 'MyNewApplication'), it's menu-entry (e.g. 'Multimedia/Audio Player') in the K-Menu (the 'start-menu') and possibly a lot of other information.

This makes it un-necessary to link to a library in order to instantiate the classes within because KDE can do it for you, and an application doesn't have to know about any header-files from that library. Only the interfaces (pointer-handles) coming out of the library needs to be known by both participants. By using interfaces commonly known (e.g. only existing Qt- and KDE-classes), many interfaces become clean enough to make Qt- and KDE-APIs the only dependencies of the main-application and the parts - they don't have to know each other directly.

Problems with the KDE/Qt solution Nothing is perfect and our solution does have some problems:

String-based type-control As part of the solution of the name-mangling-problem discussed above, the Qt-library have an base-object (QObject) which holds the type-name of a class as a string. This enables Qt to query and cast classes using strings instead of using C++'s *dynamic-*, *static-* and *reinterpret-_cast<>()*. Based on the string identification straight C casting can be done - like this : *Foo *ptr_New = (Foo *) new Bar();*.

Decentralised program-control-flow The QObject with its dynamic string-based casting, allows for a Qt-speciality to operate: the *Signal-and-Slot*-mechanism. Instead of event-ids being parsed around by C-interfaces .
But, it sometimes make it a bit unclear when some instance-function gets called - and you can try and make the call in the code, but it may not go though at run-time, e.g. because you are calling on the wrong class. But you get told at runtime - by a debug-statement on stderr.

Dynamically Shared Objects (DSO) The dynamic library loading scheme of KDE is string-based: So all strings, function-declarations, naming and placement of libraries must match up! The strings must be case-correct at times, and may automaticly pre-/post-fix themselves with "lib", "kcm_", ".la" and so on. The factories and the straight-C function-declarations ('void *init_*name*()', 'void *create_*name*()', etc) must match names from the makefile-system and to the '.desktop'-files. Placement paths for installs and makefile-target-names must be aligned with naming inside and of '.desktop'-files.

All of the above items boils down to one point: Because of name-mangling problems between compiler-versions and compilers, a lot of linkage and casting problems have to resolved at runtime by code and not the compiler. The result is that C++'s type-safety-features must be relaxed, deferred and postponed til run-time. And, that it will get resolved by custom-based code (in our case: our application and KDE using strings) - and not by the compiler.

So, making a typo-error in an string-based identifier wouldn't make the compiler slap your fingers, and nor would it necessarily break the code at runtime either - but obviously it wouldn't perform either, it just wouldn't do the request. Likewise, an error in a .desktop-file-naming or string will leave you baffled, because the library and/or it's class-factory doesn't get identified correctly. Reading the reference- and user-documentation obviously didn't do the job, so only the Qt- and KDE-source-code are available for tracing the problem-roots. And debugging isn't an option, unless you are running on a debug-version of Qt and KDE-libs - and some setup-effort *is* required to meet that requirement, and this author didn't know how to do that - in a late night flash debugging frenzy.

7.2.3 Summary

We *need* the dynamic aspect, so we will use what ever solution some component-library or OS can come up with. The KDE-solution is far superior to native straight-C-linker-loader from the Linux-OS. All though the KDE-solution is string-based in it's resolvement of linkage-points, our infra-structure will not suffer extensively, since our data-core - the SQL-database - is also string-based in it's I/O (by it SQL-nature).

Result was, that a lot of time was spent on these issues, but they deliver the desired features: Independently built- and released-components.

7.3 Defined Parts of the main system

Now having a system for plugins, it is time to define which parts exists. The individual parts from the system overview have their design-issues discussed here. We will elaborate on each part in the following sub-sections, but first a list of the parts and the compilation-modules they reside in:

lpfw The main application (KApplication) and main frame (KMdiMainFrm). It also holds the base-class for all plugin-parts to derive from. The base-class hold essential pointers like the database-connection, KDE-related global data (KGlobal) etc.

lpfw_part The support GUI for the main application and frame, like a common framework for configuration dialogs (the settings-singleton and KSettings::Dialog), shortcut-key-configuration-dialog, plugin-activation (KSettings::ComponentsDialog) etc.

lpfw_sql_db_part Housing the connection (QSqlDatabase) and widget (QDataTable in a KMdiChildView) to the database-backend. This is the **DBView**.

lpfw_ipt_cmd_part Manages conversion of data between the database and iptables-save/-restore commands (using a KProcess to manipulate and stream standard-in-, -out-, and -error-handles to the iptables-commandline-process). This is the **RuleView**.

lpfw_proc_view_part Parses information of processes and their sockets running on the host to the database (using KProcess to manipulate a lsof-commandline-process). This is the **ProcView**.

lpfw_netstack_part Parses information of the NICs found on the host and the routing they will perform - i.e. the network-layout seen from the host. NOT created as an separate part, but housed inside the lpfw_proc_part - for now. It also holds the SetupWizard for getting started with some rules with your particular NICs and network-layout.

Several *future* modules would feasible right now and be quite easy to fit in:

lpfw_susefirewall_part Holds a view of the variables of the SuSEfirewall-script (/etc /sysconfig /SuSEfirewall2) which sets up the firewall by executing iptables commands on the fly, based on the variable-values. Rather independent for the moment, since it executes in a monolith block of commands, but it does generate iptables commands (which are post-editable) and it is a good (bash-based) setup-wizard.

lpfw_firehol_part Holds a view and editor using the FireHOL-language. This is a simplified language for setting up a firewall more easily. It's not a wizard and a monolith block execution (like SuSEfirewall) of bash-scripted iptables commands, but still post-editable and can also be used as a quick-setup part.

lpfw_idiot_part A informal sanity checker, that protects the user from shooting him self in the foot. Ensuring that useful rules are preserved - like allowing DNS-traffic or loopback-traffic on the firewall, so that e.g. the XWindow-system can function - and hence, KDE and this application can be seen at all. Formally, it isn't illegal to remove such 'I-don't-know-what-this-does-so-its-probably-unnecessary' rules - but usually it makes the user feel stupid afterwards, unless the user is an expert.

In general:

lpfw_XXX_part Any 3. party development of new views (reflections on relations in the database), new checkers/verifiers - i.e. other peoples triple-X-rated material (security wise).

All parts are easily created by using the *kapptemplate*[1]. It comes standard with newer KDE-developer installations and make a KPart-project in a flash.

7.3.1 The main application frame

The application is based on the well-known MDI-structure known from the windows world and seen in KDevelop, Konqueror, KOffice etc. It holds the application object (KApplication) containing the entry/exit-point (the main()-function), XWindow-event-handlers (event-dispatch-loop), config-information (KGlobal/KStandardDirs/...) and the main widget (KMdiMainFrm) holding the GUI-Frame with the title-, status-, menu- and tool-bars.

Each part is a ordinary KPart creating a QWidget. It is created by the KDE-factories and wrapped within a tabbed widget (KMdiChildWidget), so it has no knowledge about MDI-structures etc. The KParts-technology (XML-GUI-merging) handles building widgets, menus and toolbars for the parts and merge them into the application. The KCM's caters for the configuration that any parts may have - they plug in to the main config-page of the application.

The main application *will not* define or demand a common interface-base-class, which all parts for this application must use (i.e. know header-files and link to). This can be allowed, because of the KDE-part-technology, where parts can know each other and only the ones they need to. This prevents:

1. Base-class explosion of an interface that must cater for several un-related parts - or that multiple interfaces have to be defined.
2. The main application doesn't do much processing it self - so it would only be serving as a carrier of information, parsing pointers between parts defined in the interface. Such is not needed, since the parts can know eachother and just define some local interfaces between them as necessary.

[1]From the man page: kapptemplate is a shell script that will create the necessary framework to develop various KDE applications. It takes care of the autoconf/automake code as well as providing a skeleton and example of what the code typically looks like.

3. Since parts are build independently, parts can more easily be expanded with special interfaces towards other parts later on.

4. Parts-in-parts can combine, simplify or conceal interfaces. Such can be thought of as wrapping one part up in another part (i.e. an interface-adaptor-part).

An core example of the above is the database-part. It has a *create-me-a-connection*()-interface returning an instantiated QSqlDatabase-pointer (from the Qt-SQL-Module-library). With it, *any* part can make transaction-safe SQL-queries using the Qt-class QSqlQuery (e.g '*QSqlQuery(QSqlDatabase *, "SELECT * FROM foobar");*') - all without knowing database-type, -host, -username or -password. All other parts (and the main application) is linked against Qt/KDE-libs and hence doesn't have to link to our datatbase-part-library, in order to get the QSqlDatabase-pointer.

Apart from that interface, the database-part contains another interface used by the main application. It creates a fully fledged widget with a editable table-view of all the database-tables. But most other parts don't kneed to know that interface - only the main application have this need, and it may evolve independently (build-wise) without breaking inter-part-comparability.

Additionally, the DCOP-interfaces allow a C/C++ and script-based access, which isn't link-dependable either.

7.4 The Database

Finding out that a database was needed and choosing a database engine turned out to be time consuming, since I knew nothing about SQL - or for that matter anything about database-theory and -practice. My supervisor pushed a good book[DBSystems, Chap 1-8] on me which took care of most details on theory and practice was found on the [ERD-Intro]-site.

7.4.1 Choosing a Database-engine

Initial choice was MySQL, because Entity-Relation diagrams could be drawn by the program *gerwin*. It can not only draw the ER-diagrams, but also show the resulting SQL-Tables and generate the SQL-commands for making the tables in MySQL. Also, MySQL have a good proven record, it's fast, has ease-of-use with GUI-editor, and can be embedded easily. But it had bad docs compared to Postgres, had far less SQL-standard-support (no sub-queries, stored procedures/functions, very little constraints enforcement), a much less optimised C-API, which also seemed more difficult to use.

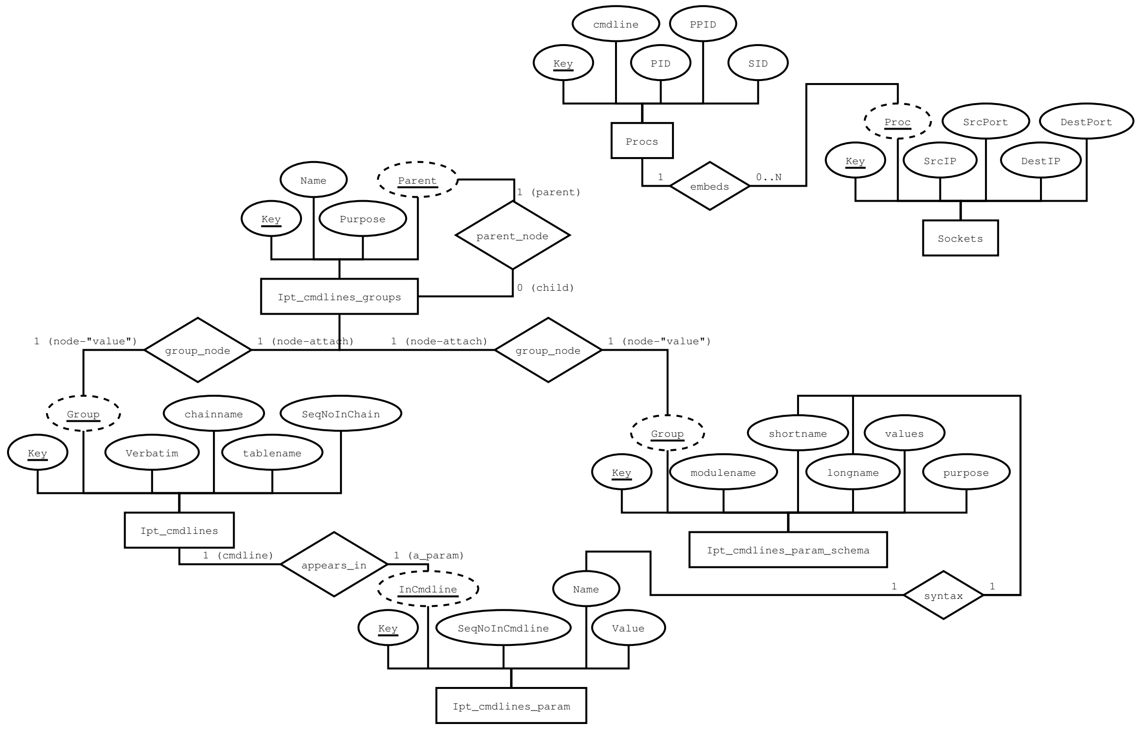
As more was learned about SQL in general and about MySQL and Postgres, it became clear that MySQL would require more programming down the line, because it have fewer possibilities across the board. MySQL have less support for e.g. searching using regular expressions, sub-SQL-queries, stored procedures (stored queries and views) or foreign-key-handling-constraints like CASCADE-ON-DELETE. That means it'll have to be done by the application-code instead, and that's just an extra unnecessary load.

By switching to Postgres as database backend, the issues named with MySQL are resolved, though Postgres is larger, more complex and slower.

To access Postgres from the C++-code, the *pqxx*-library was initially chosen, since it has a C++-typesafe API (by using C++-templates and STL), though it lacks any GUI-aid. But Qt has better overall ease at handling any data. It also have separation between Database-backends (Oracle/Postgres/MySQL/etc.), SQL-handling (Connection-, transaction- and result-classes) and GUI-presentation (edit-fields, list- and tree-views, other widgets).

7.4.2 The ER-model

Our data have to be laid out into database-tables. We have strived to make the ER-model adherent up to 3rd normal-form, and been using *dia* to make the diagrams.



(a) Note: Stippled oval attributes are the foreign-key-relations in tables.

Figure 7.1: ER-model overview..

The two tables in the top (*procs* and *sockets*) aren't in the SQL-database, they are implemented as lists in memory. Since they are changing from session to session, there isn't a point saving them to disc for re-establishment. The general interface of SQL allowing other-modules to access the information really isn't used, since only the ProcView uses them for now. So, they remain inside the ProcView as lists in memory for now.

The four main tables in the database are:

Ipt_cmdlines Holds the rules (one line per entry), grouped per chain.

Ipt_cmdlines_parameters Holds parameters of rules, grouped per rule.

Ipt_cmdlines_parameters_schema Hold all the possible parameters the parser can meet on the commandline.

Ipt_cmdlines_groups Creates a tree-like structure of parent-child-nodes. Chains, tables, configurations etc. only exists as a group-node, since they are all very simple: they have a name.

The glue is the *Ipt_cmdlines_groups*-table. It make it possible to create trees as seen in the listviews of RuleView and ProcView. The tree is a straight forward *parent-with-children-nodes*-tree. Each parent-node have a (primary) key, to which each child-node refers to - i.e. children knows who their parent are, and a child can only have one parent - see Fig. 7.2 for an actual example.

Entries in the other tables (*Ipt_cmdlines_*XXX) are also linked by foreign-key . Each parameter-entry in the Ipt_cmdlines_parameters will link to the node of the rule it belongs to in the Ipt_cmdlines_groups. Each rule-entry in Ipt_cmdlines will link to the (rule-)node, and that node will link to the config-node etc.

This lend it self well to the fact that a parameter is attached to one rule (only), a rule to a chain, chain to table, table to config, config to host, etc. Just about anything can link to a node in the tree and become part of that cluster or group of information.

This makes it very flexible and adaptable for future additions of data we don't know of at the time of compile and release. Other modules can hook up information to configurations, chains, rules, parameter etc. as they see fit, without extending or changing our table-definitions, and it will not affect our algorithms, since we'll choose to propagate on SQL-DELETE and -UPDATE (foreign-key issue), so we don't leave any dangling ophans in the database.

7.4.3 The Postgres Table definitions

Below are the resulting definitions to make the tables in Postgres. The *global_seq_number_roll* is the unique key-generator, from which all new keys are drawn from. Otherwise most values are text-based, since it is most flexible.

Foreign keys do have restrictions: parameters must exists in the schema, for a particular parameter to parsed. Likewise the rule must exist before we can hook up any parameters on it, configurations must exists before rules can be appended etc. The reverse is also true on deletion: When the configuration is deleted, all rules and parameters are deleted too (i.e. no orphans).

```
-- for globally unique serial-ids
CREATE SEQUENCE global_seq_number_roll;
---------------- make the table definitions ---------------------
-- holds the group dependencies of iptables-cmdlines-groups (tree-structure)
CREATE TABLE Ipt_cmdlines_groups (
        key INTEGER UNIQUE DEFAULT nextval('global_seq_number_roll') NOT NULL,
        type TEXT,              -- function/type of this node ('a chain-type')
        name TEXT,              -- basic name-tag ('call-sign', 'anti-spoof-ip')
        purpose TEXT,           -- like: ('rootnode', 'base node for hostX', 'eth1')
        group_id INTEGER,       -- this group depends on the parent (rule->chain)
        PRIMARY KEY (key),
        FOREIGN KEY (group_id)      -- can be self-loop for a rootnode (no parent)
                REFERENCES Ipt_cmdlines_groups (key)    -- parent group must exist
                ON UPDATE CASCADE       -- track parent group
                ON DELETE CASCADE       -- unlink dependency to parent group first
);
-- holds possible parameters that can be meet on the cmdline
CREATE TABLE Ipt_cmdlines_parameters_schema (
        key INTEGER UNIQUE DEFAULT nextval('global_seq_number_roll') NOT NULL,
        module TEXT,            -- the module the paramater belongs to ('state')
        shortname TEXT,         -- short options name ('-s')
        longname TEXT,          -- long options name ('--short')
        values TEXT,            -- possible values ? ('1-1024')
        purpose TEXT,           -- helpful explaination
        group_id INTEGER,       -- several commands may belong to a group
        PRIMARY KEY (key),
        FOREIGN KEY (group_id)
                REFERENCES Ipt_cmdlines_groups (key)    -- parent group must exist
                ON UPDATE CASCADE       -- track parent group
                ON DELETE CASCADE       -- unlink dependency to parent group first
);
-- holds the actual iptables-cmdlines
CREATE TABLE Ipt_cmdlines (
        key INTEGER UNIQUE DEFAULT nextval('global_seq_number_roll') NOT NULL,
        verbatim TEXT,                  -- actual line parsed
        configname TEXT,                -- the config it belongs to
        tablename TEXT,                 -- the table it belongs to
        chainname TEXT,                 -- the chain it is/belongs to
        seq_no_in_chain INTEGER, -- chains: NULL-values, rules: seqential numbered
        group_id INTEGER,               -- several commands may belong to a group
        PRIMARY KEY (key),
        FOREIGN KEY (group_id)
                REFERENCES Ipt_cmdlines_groups (key)    -- parent group must exist
                ON UPDATE CASCADE       -- track parent group
                ON DELETE CASCADE       -- unlink dependency to parent group first
);
-- holds actual parameters meet on the cmdline
CREATE TABLE Ipt_cmdlines_parameters (
        key INTEGER UNIQUE DEFAULT nextval('global_seq_number_roll') NOT NULL,
        appears_in_cmdline INTEGER,     -- foreign key link to the cmdline
        seq_no_in_cmdline INTEGER,      -- seq-number in that cmdline (4th?)
        from_module INTEGER NOT NULL,   -- from which module(e.g. '--set-mark mark'
                                        --            from '-m mark' or '-j MARK')
        name TEXT,                      -- name ('--jump')
        value_t TEXT,                   -- value ('ACCEPT')
        group_id INTEGER,               -- several parameters may belong to a group
        PRIMARY KEY (key),
        FOREIGN KEY (group_id)
                REFERENCES Ipt_cmdlines_groups (key)    -- parent group must exist
                ON UPDATE CASCADE       -- track parent group
                ON DELETE CASCADE,      -- unlink dependency to parent group first
        FOREIGN KEY (appears_in_cmdline)
                REFERENCES Ipt_cmdlines (key)           -- the cmdline it appears in
                ON UPDATE CASCADE
                ON DELETE CASCADE,
        FOREIGN KEY (from_module)
                REFERENCES Ipt_cmdlines_parameters_schema (key) -- the param's module
                ON UPDATE CASCADE       -- if module changes name etc. just update link
                ON DELETE RESTRICT      -- don't allow delete of module while in use
);
```

When populated with a little test data the four tables look like Fig 7.2.

The interesting numbers in Fig 7.2 are the *key* and *group_id* of table *ipt_cmdlines_groups*. Notice how children link from *group_id* to *key* of their parent. The *key* of *ipt_cmdlines_groups* is used in the other tables too - in their *group_id*-columns. So the scheme is: Other tables link to a node in *ipt_cmdlines_groups*, which then links to a parent-node in itself.

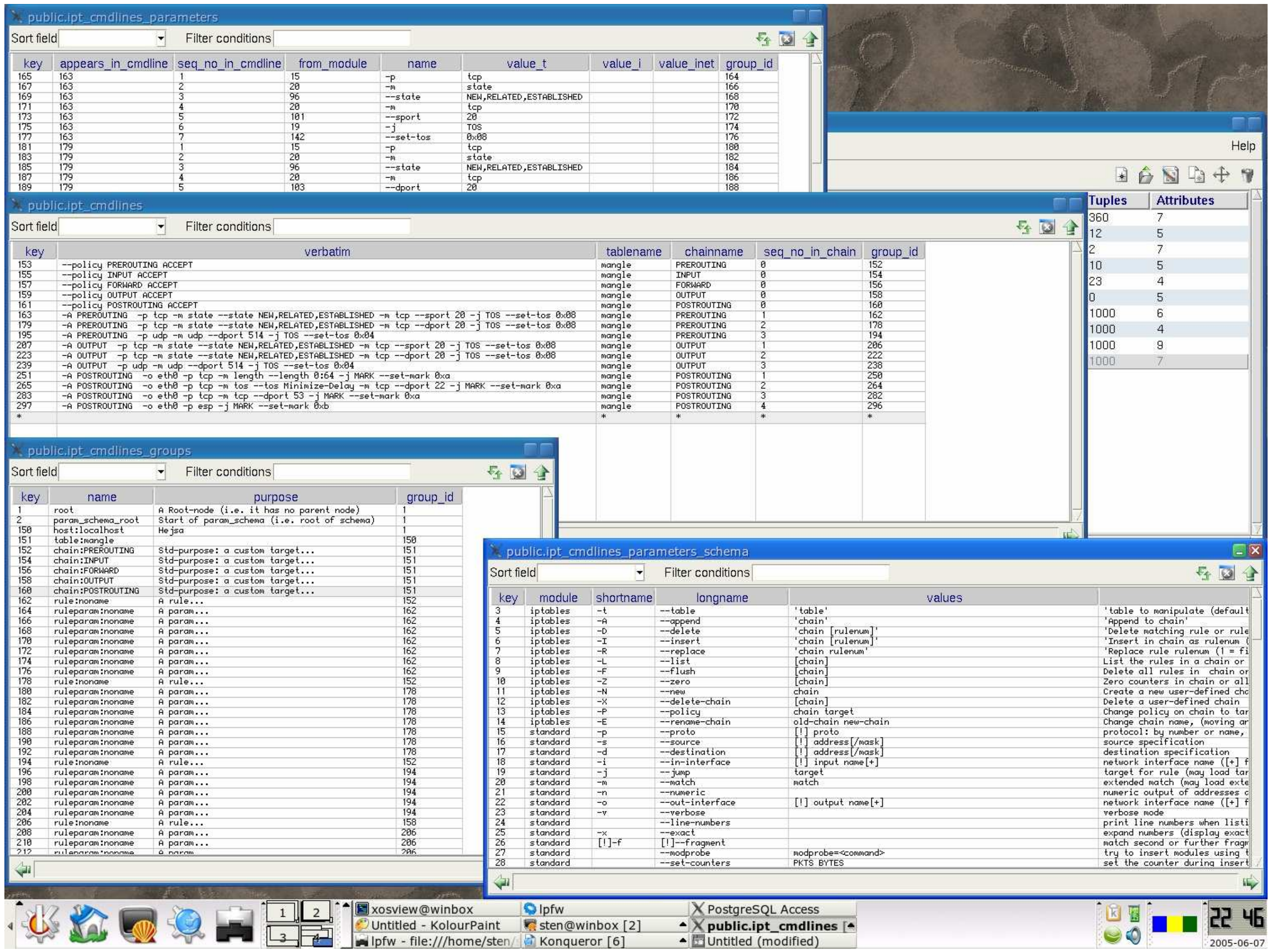
7.4.4 The database-setup

The database is using a ordinary straight forward Postgres-connection to some Postgres-server, running on the local (or a remote) host. The connection-details are in the config-page for the database-module[2], but in order to connect, the Postgres-server *must* be installed (e.g. using RPM-packages) and configured to accept connections and with the desired user(s) and password(s).

<u>**So**</u>: You won't get up and running without the following Postgres-fiddling:

- Installing is done using rpm's - see the distribution's documentation for howto.
- Configuring is done by reading the Postgres-docs.
  In short:
  - General: Enabling non-localhost connections (IPv4-sockets) to the database:
    You need to add an authentication-line to the *pg_hba.conf*-file. See "Chapter 19. Client Authentication" for how to do it. But appending the following line to the file, did it for me:
    'host all all 192.168.100.101/24 trust'
    The line makes the Postgres-server accept IPv4-connections (host) from any host within that subnet (192.168.100.101/24) - by any username (all), to any database within (any) with no password asked (trust)!.
  - General: Using non-local connection-usage (IPv4-sockets) to the database:
    <some_command> –host=host –port=5432 –username=myNewUser' [other_options...]
    The password will prompted for - if set/necessary. It can be given on the commandline

[2]Note: The connection <u>is not encrypted</u>, and connection details (password and username) <u>are stored in clear-text</u> in the config-files.



public.ipt_cmdlines_parameters

key	appears_in_cmdline	seq_no_in_cmdline	from_module	name	value_t	value_i	value_inet	group_id
165	163	1	15	-p	tcp			164
167	163	2	20	-m	state			166
169	163	3	96	--state	NEW,RELATED,ESTABLISHED			168
171	163	4	20	-m	tcp			170
173	163	5	101	--sport	20			172
175	163	6	19	-j	TOS			174
177	163	7	142	--set-tos	0x08			176
181	179	1	15	-p	tcp			180
183	179	2	20	-m	state			182
185	179	3	96	--state	NEW,RELATED,ESTABLISHED			184
187	179	4	20	-m	tcp			186
189	179	5	103	--dport	20			188

public.ipt_cmdlines

key	verbatim	tablename	chainname	seq_no_in_chain	group_id
153	--policy PREROUTING ACCEPT	mangle	PREROUTING	0	152
155	--policy INPUT ACCEPT	mangle	INPUT	0	154
157	--policy FORWARD ACCEPT	mangle	FORWARD	0	156
159	--policy OUTPUT ACCEPT	mangle	OUTPUT	0	158
161	--policy POSTROUTING ACCEPT	mangle	POSTROUTING	0	160
163	-A PREROUTING -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 20 -j TOS --set-tos 0x08	mangle	PREROUTING	1	162
179	-A PREROUTING -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j TOS --set-tos 0x08	mangle	PREROUTING	2	178
195	-A PREROUTING -p udp -m udp --dport 514 -j TOS --set-tos 0x04	mangle	PREROUTING	3	194
207	-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 20 -j TOS --set-tos 0x08	mangle	OUTPUT	1	206
223	-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j TOS --set-tos 0x08	mangle	OUTPUT	2	222
239	-A OUTPUT -p udp -m udp --dport 514 -j TOS --set-tos 0x04	mangle	OUTPUT	3	238
251	-A POSTROUTING -o eth0 -p tcp -m length --length 0:64 -j MARK --set-mark 0xa	mangle	POSTROUTING	1	250
265	-A POSTROUTING -o eth0 -p tcp -m tos --tos Minimize-Delay -m tcp --dport 22 -j MARK --set-mark 0xa	mangle	POSTROUTING	2	264
283	-A POSTROUTING -o eth0 -p tcp -m tcp --dport 53 -j MARK --set-mark 0xa	mangle	POSTROUTING	3	282
297	-A POSTROUTING -o eth0 -p esp -j MARK --set-mark 0xb	mangle	POSTROUTING	4	296
*		*	*	*	*

Tuples	Attributes
360	7
12	5
2	7
10	5
23	4
0	5
1000	6
1000	4
1000	9
1000	7

public.ipt_cmdlines_groups

key	name	purpose	group_id
1	root	A Root-node (i.e. it has no parent node)	1
2	param_schema_root	Start of param_schema (i.e. root of schema)	1
150	host:localhost	Hejsa	1
151	table:mangle		150
152	chain:PREROUTING	Std-purpose: a custom target...	151
154	chain:INPUT	Std-purpose: a custom target...	151
156	chain:FORWARD	Std-purpose: a custom target...	151
158	chain:OUTPUT	Std-purpose: a custom target...	151
160	chain:POSTROUTING	Std-purpose: a custom target...	151
162	rule:noname	A rule...	152
164	ruleparam:noname	A param...	162
166	ruleparam:noname	A param...	162
168	ruleparam:noname	A param...	162
170	ruleparam:noname	A param...	162
172	ruleparam:noname	A param...	162
174	ruleparam:noname	A param...	162
176	ruleparam:noname	A param...	162
178	rule:noname	A rule...	152
180	ruleparam:noname	A param...	178
182	ruleparam:noname	A param...	178
184	ruleparam:noname	A param...	178
186	ruleparam:noname	A param...	178
188	ruleparam:noname	A param...	178
190	ruleparam:noname	A param...	178
192	ruleparam:noname	A param...	178
194	rule:noname	A rule...	152
196	ruleparam:noname	A param...	194
198	ruleparam:noname	A param...	194
200	ruleparam:noname	A param...	194
202	ruleparam:noname	A param...	194
204	ruleparam:noname	A param...	194
206	rule:noname	A rule...	158
208	ruleparam:noname	A param...	206
210	ruleparam:noname	A param...	206

public.ipt_cmdlines_parameters_schema

key	module	shortname	longname	values	
3	iptables	-t	--table	'table'	'table to manipulate (default
4	iptables	-A	--append	'chain'	'Append to chain'
5	iptables	-D	--delete	'chain [rulenum]'	'Delete matching rule or rule
6	iptables	-I	--insert	'chain [rulenum]'	'Insert in chain as rulenum (
7	iptables	-R	--replace	'chain rulenum'	'Replace rule rulenum (1 = fi
8	iptables	-L	--list	[chain]	List the rules in a chain or
9	iptables	-F	--flush	[chain]	Delete all rules in chain or
10	iptables	-Z	--zero	[chain]	Zero counters in chain or all
11	iptables	-N	--new	chain	Create a new user-defined cha
12	iptables	-X	--delete-chain	[chain]	Delete a user-defined chain
13	iptables	-P	--policy	chain target	Change policy on chain to tar
14	iptables	-E	--rename-chain	old-chain new-chain	Change chain name, (moving ar
15	standard	-p	--proto	[!] proto	protocol: by number or name,
16	standard	-s	--source	[!] address[/mask]	source specification
17	standard	-d	--destination	[!] address[/mask]	destination specification
18	standard	-i	--in-interface	[!] input name[+]	network interface name ([+] f
19	standard	-j	--jump	target	target for rule (may load tar
20	standard	-m	--match	match	extended match (may load exte
21	standard	-n	--numeric		numeric output of addresses c
22	standard	-o	--out-interface	[!] output name[+]	network interface name ([+] f
23	standard	-v	--verbose		verbose mode
24	standard		--line-numbers		print line numbers when listi
25	standard	-x	--exact		expand numbers (display exact
26	standard	[!]-f	[!]--fragment		match second or further fragm
27	standard		--modprobe	modprobe=<command>	try to insert modules using t
28	standard		--set-counters	PKTS BYTES	set the counter during insert

Figure 7.2: Tables filled with test-data (pgaccess).

too, but then it can be see in clear-text by by any other user by doing a 'ps'- or 'top'-command! - so don't do that.

  – General: To create a user for a database (owner of the database and -tables): 'createuser myNewUser'
  – General: To create a database for all the data: 'createdb –owner=myNewUser mydb'
  – The easy way using some default-settings (From the Docs: "By default, a database with the same name as the current user is created."):
    'createuser lpfw ; createdb –owner=lpfw lpfw ; psql -h host -U root_user'
    and within the 'psql'-terminal issue a: 'ALTER USER lpfw WITH PASSWORD 'lpfw'' - and you're done. A user ('lpfw') with password ('lpfw') have a database ('lpfw') is created and a connection is established (can be exited with '\q'). Now the connection can be tested with:
    'psql -h localhost -U lpfw -d lpfw'
  – More usage and help can be see in section 'I. Tutorial' and 'III. Server Administration' of the Postgres-docs, and by looking at the man pages or usage-help (issuing the –help option) of the commands in question. Most of the setup-commands require root-access to the database-server.

- Set the connection-setting in the database-config-page to the created values to enable usage.

Some systems (e.g. SuSE) have a Postgres-startup-setting-file in '/etc/sysconfig/postgres', where you should add '-i' to the startup-args, to get Internet-sockets enabled. Finally, if applicable, don't forget to open[3] up your firewall on port 5432.

7.5 The GUI-Views

The GUI-framework is build using Qt/KDE. It consists of the well-known model of Document/View-architecture. The Document-model is the data managed by the database-backend. The Views are MDI-Child-frames constructed as KDE-KParts. This results in a familiar user-interface recognisable to most users and maximising Qt/KDE-component-usage, since a vast range of classes exists to support such layout.

7.5.1 Experience using KDE and Qt

Some patterns emerged during the development cycles with KDE/Qt. First, one would read the Qt-documentation and then the KDE-documentation, for classes and topics which overlap. The overlapping topics are many, since KDE have many Qt-derived classes, and Qt in turn have implemented many KDE-invented technologies from the KDE-libraries back into the Qt-libraries. E.g. both offers XML-based dynamically loadable dialogs, part-shared libraries, DragN'Drop, GUI-classes for List-views, Dockable main-application-widgets etc.

The Qt-documentation is good and elaborate with both user-guides, howtos, examples and sourcecode. The KDE documentation is sometimes present and fulfilling, with respect to user-guides and howtos, but can be wrong, outdated and scattered. The KDE reference documentation is browsable by sourcecode and depends on how good the programmer documented his code.

First, one would implement the way Qt advised - using the KDE-derived classes. Then one would find out, that the Qt-way works, but that the KDE-derived classes already have done most of the work and offers more elevated functionality. The KDE classes would then offer a slightly changed interface and way of implementation, for achieving the same functionality as Qt.

So, first you do it the Qt way - and find out, it's the wrong way (but it works as Qt says). By then, you have gained insight in the mechanics of the topic, which then enables you to understand

[3]An intractable problem: how do you do open up a port in your firewall? . . . with our program you're about to install?

the KDE-documentation and -sourcecode. That make you realise how the KDE way is supposed to work, and then you do it the second time around - the KDE way - which is the right way.

The derived conclusion is: Frustrations arise, as it takes twice the time to figure out the usage of some KDE-class - but you get more functionality, code reuse, tested code, constancy in layout etc. So, it's worth the effort.

7.5.2 Debugging and testing GUI-code

It takes a lot of testing to get complex-GUI to look easy and right. Testing is notoriously difficult to automate, and in this case it was done manually (by hand), by simply using the GUI.

When encountering problems, debugging may not always be available e.g. as when DragN'Drop-problems occur. Because it's an interactive process, it isn't always an option to halt the program with the debugger - as it is in a parallel execution with the GUI-desktop. The desktop is running the XWindow-message-pump, which is the code at the other end of the DragN'Drop operation - and it *doesn't* like to be halted. Trying to, makes the XWindow-cursor and input-system do weird stuff and sometimes freeze.

So, the only option is the old fashioned debugging approach: printf()'s - and loads of it. Since we got a log-window we utilise it for all our debugging needs by printing all sorts of internal data on to it, when enabled. The Verbosity-level 3 is the enabler - except when errors occur: they are level 1.

7.5.3 The RuleView

The RuleView is shown in Fig. 6.3, and is implemented by deriving from KDE's KListView- and KListViewItem-classes. The idea is that each line in the view is an item, so instances of listviewitems are added to the listview. The reason we need to derive is to overload the virtual functions for clicking and DragN'Drop-support. The Qt-documentation [KDERef] have howto-sections on both views and DragN'Drop.

Several problems arose during implementation, listed below in random order.

No cloning of items The listviews and -items aren't properly separated, they are friends of each other (really intimately coupled). Probably therefore, Qt have disabled the copy-constructor and assignment-operator on the item-class, because a listview and it's items cannot deal with a cloned item.

The implication is that having two rule-views side-by-side for e.g. DragN'Drop becomes much more troublesome, since we need to keep track of two items (one in either view) and transfer data from one to the other - but we cannot assign ('=') or copy the items. We have to write deep-copy transfer-functions, that clones most of the item's member-variables. It proved to be error-prone and isn't working right now - but of course, it can be made to work.

The copying also affects DragN'Drop: The dragged object (the item) is cloned, because the system takes over ownership of a drag-object. The system then *deletes* it when appropriate, so we cannot give the system our item - it must be a disposable copy of it. We use a text-string of our *sortkey* (see sorting issues in the listview below). The clipboard uses the same mechanism as DragN'Drop, so we don't support that very well either (you'll just get a text-string representing the columns of the item).

Referencing a particular item as *the* dragged item was troublesome. Since we cannot make an exact copy of an item, the comparison-operator ('==') becomes troublesome, because we can't uniquely refer to a specific item by comparison, since they will not be alike on some particular member-fields - only on most of them. And we can't use the item's reference ('&') - i.e. the pointer-address as unique identifier either, because then it gets deleted by the system later on...

Irregular sorting of items The listview does not sort its items before they become visible (by scrolling or unfolding). Sorting is important, because the order of the rules *are* important and isn't simply by alfa-numeric order.

On the other hand, just using the data-values of each item (e.g. the *seq_no_in_chain*) is problematic, because not all items are in the database yet and do not have these values initialised yet. Instead, we have to iterate though the view's items, to get the exact number for e.g. *seq_no_in_chain* for each rule, in each chain (in each configuration…). Auto-numbering the items sequentially and use that number as sorting - is troublesome too: As the insert- and delete-operations requires/makes number-holes in the sequence, and we *must* avoid having them jump around as we read them in or edit them, so auto-numbering must be strictly adhered.

The sorting *does* work, but it was really painful to get it in that state. We have invented our own internal sortkey that is sequentially numbered to fixate the items during edits, except when we initially populate the view from the database. There, we use the creation dates of the nodes (lowest primary keys) and seq_no_in_chain/seq_no_in_rule - depending on the type of item (config, table, chain, rule, param etc.).

Slow database updates The listview was intended to use the SQL-database and it's strong search, selection and ordering facilities. However, the turnaround-time for a single query is in the hundreds of milliseconds (200-400 mSec), which is too slow for interactive DragN'Drop-operations and editing. When dragging, we must have 25 frames/sec for a smooth scrolling window with snappy updates. But that means the turnaround-time must be less than 1/25 Sec. (i.e. <40 mSec.) - and the problem only gets worse as the database grows.

The solution was to make the listview a cache of the database - it works just like a write-through-cache to the SQL-database. That is, read the database, populated the view, do our edits, and then done writes back our changes. Meanwhile the database better not change, i.e. get some keys detected, entries renamed etc. And locking the tables is undesirable, since the database is intended to be an interface to several modules for independent concurrent access.

The database turnaround-time is well suited for popping up a dialog of an item to edit, or for large bulk editing - but not for DragN'Drop.

The net result of the above issues, is that the listview have become a write-through-cache of the SQL-database. Which means we had to implement the synchronising, searching and sorting mechanisms, that otherwise was intended to get the SQL-database to do for us. We winded up having mirrored at lot of the basic functionality of a database (even including types…).

Solution to the cache-problem The database-solution implemented right now is relying on the locking mechanism. But the real solution is to implement the state-machine of Fig. 7.3 for an item in the listview.

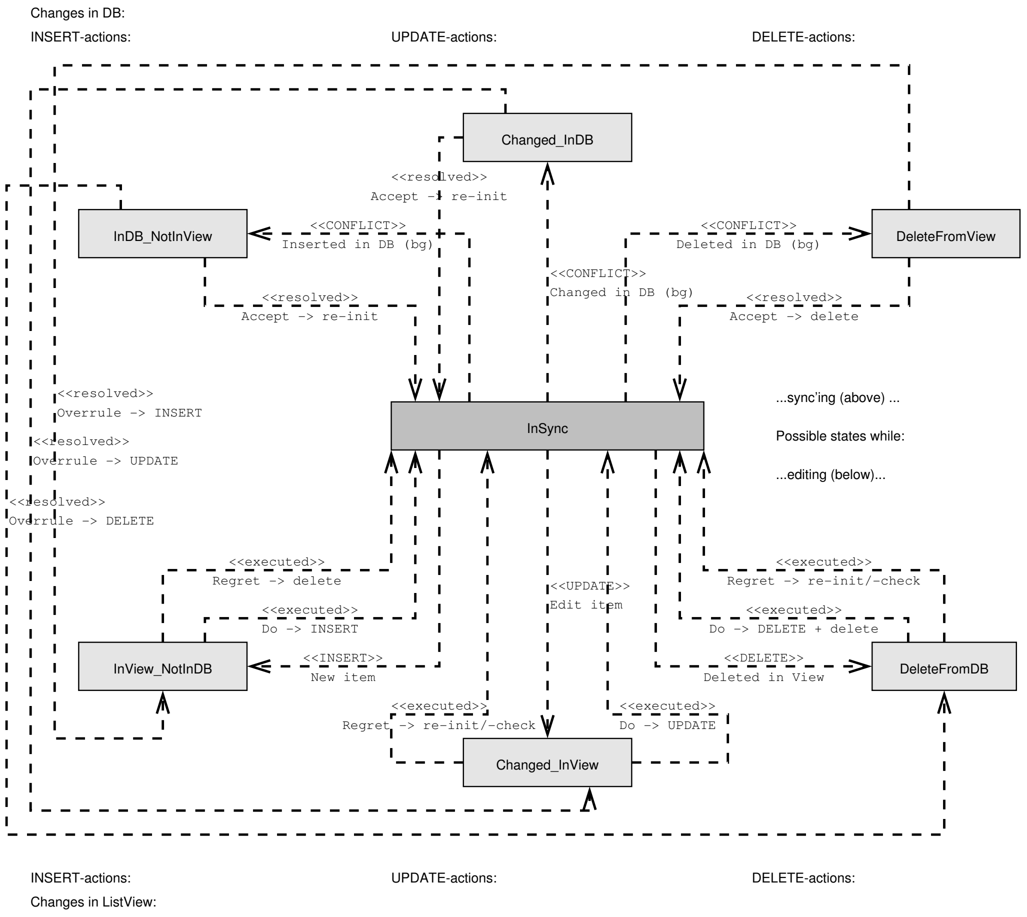


Figure 7.3: States for an ListViewItem .

The states comes in groups of three's: Database-changes (top), Listview-changes (bottom). Changes are insert (new), update (modified) and delete (removed). We'll refer to listview-data as *items*, and the corrosponding database-data as *entries*.

All items initially start out InSync, as they are read in from the database and populates the view. Once editing begins, an item will switch to one of the bottom-states, depending on the action of the user. To get out of one of the changed states and get back to InSync, we must either commit changes from listview to database (by using SQL-INSERT, -UPDATE or -DELETE commands), or regret the changes by re-initialise the item from the database (or simply delete the item in case of a deleted entry). This constitutes normal straight forward operations of editing in the listview and then committing the changes back into the database.

The picture get more complicated when the database isn't locked and can have changes too. During the synchronising-stage of committing listview-changes to the database, each item can be detected as being changed in it's corresponding database-entry. I.e. it can have been changed, deleted or new items can have been inserted, which didn't figure in the listview before.

We must then select which changes we want: The listview-edits or the changes in the database? Luckily, they are each others reverse, so we can choose freely. Since an item, which have been deleted in the database (i.e. the entry is gone), the item's state simply switches (from Delete-FromView) into the new-in-listview-state (InView_NotInDB) - in order to revert the database-change. Likewise, we can revert a newly inserted entry in the database (InDB_NotInView), by marking it as a delete-item in the listview (DeleteFromDB). And finally, changed-in-listview and changed-in-database are each others complements, just select which one to resolve the conflict.

All the top-states then becomes conflict-states of items that have changed in their database-entry, and needs to be resolved by either accepting the database-changes (by discarding the listview-items) or reverting the changes (by imposing the listview-values back into the database-entries). The listview holds a time-snapped branch of the database, the newly discovered changes in the database holds the other branch. The conflict is resolved by selecting the branch-parts we want.

The process of the synchronising-stage then get expanded to 1) lock the tables, 2) scan for changes in the database-entries, 3) resolve conflicts, 4) commit (or regret) changes from the listview to the database with the SQL-commands. That ought to do it, and it carries some interesting properties with it. It now becomes possible to:

- Keep the database-tables unlocked most of the time, while editing or browsing. Only during synchronising (populate and commit) the listview with the database, are the tables locked (for writing).
- It becomes possible to see and show the changes in the database-entries made by others modules, as they will appear as changes in the database. And by resolving the conflicts, it becomes possible to reject (or change) the changes made by those 'foreign' modules.
- It becomes possible to diff two sub-trees - say, two configurations - to see the differences (and possible do a merge) of the two 'conflicting' trees. The listview is 'set' on one sub-tree, and the listvie's internal database-pointer is simply 'set' to the other sub-tree (instead of, as normally, the corresponding entries of the listviewitems). Then, it will look like changes have occurred to the items, simply because the database entries are now pointing to another sub-tree, instead of itself (I know - tricky, get it?).

With the more elaborate diff-opportunities presented above, the RuleView shows one of it's more concealed roles: It shows changes in context - a bit like a giant dialog saying 'Do you accept these changes?'.

Solution to the item-cloning, -referencing and -sorting-problems The cloning- and irregular sorting-issues should be mostly solved by Qt-release V4. It was released in June - 2005, and KDE-porting to use V4 is underway (KDE V4). The new Qt-version promisse to resolve a bunch of issues encountered in this project, - so I figure, it can be concluded that I wasn't the only one with a problem!

The View-classes (QIconView, QListView and QTableView), along with their Item-classes have been deprecated, and are replaced by a single InterView-class (where do they get those names from...).

The old listviews are not document-model-architectures. That implies mirroring/conversion/re-sync between internal data-representation (i.e. the db-tables in our case) to listviewitems. Then presentations and manipulations of the items are done and re-converted back into the internal-data-structures.

The new InterView-class are an document-model-architecture, where instead the internal data-representation will have to conform to some interface-functions for get/set-ing of data. This way several views can share the same instance of data (and update synchronously).

From the sales-wrapping:

> *Model-View Item Views*
>
> *Qt's new Model-View architecture is designed for the display and manipulation of data pulled from any data source, with support for SQL databases. The item views are highly scaleable, and are particularly useful for maintaining efficiency and UI responsiveness while processing huge datasets in the background. Data can be presented in table, and tree-structured views. Both display and user editing are fully customizable.*

Other improvements in Qt V4 related to this application are:

- New re-factored SQL-, XML-, and Network-modules,
- New Dynamic-library-loading- too,
- The code are now structured into several libs (ala same ordering as KDE),
- They have a new dockable main-app-view (like the KDE-class we use),
- Better support for casting through shared-libs-boundaries (linkage-/mangle-issue) etc.

More information on *http://www.trolltech.com/products/qt/qt4info.html* (and in-depth technical: *http://doc.trolltech.com/4.0/qt4-intro.html*). There are many changes and improvements to issues, that this project have been subjected to (and suffered under;-).

We have to wait for the KDE-porting to finsh and release it's corrosponding version (V4). By then, the project's source code needs to be re-factored to take advantage of the new Qt V4 improvements. The RuleView is already in need of a cleanup of the code[4], and it would be practical to hit 'both birds with one stone'.

7.5.4 The ProcView

The ProcView (Fig. 6.4 on page 55) is basically the same skeleton as the RuleView - only its a lot simpler. It took about 1 week to get up and running, compared to the 6-8 weeks of struggling with the RuleView. But of course, by then most general listview-issues had already been resolved or circumvented.

The ProcView is poised for far fewer problems, since it doesn't depend that much on sorting and order, doesn't get updates behind it's back and doesn't move items around by DragN'Drop.

Time-wise, it was created a bit late and in addition, it got the network-layout-part cramped in - including the SetupWizard (See Sec. 7.8). So code-wise it is somewhat rough around the edges, and also in need of a code-cleanup/re-factoring.

7.6 Root-execution security

The installment and execution of the application should be secure.

The commands being executed in sub-shells are all commands which require root-rights. By placing them in a sub-shell, the main application can run with ordinary rights and needs to obtain root-rights, and hence it don't need to ask the user for the root-password, except for the one time setup scenario where the user must put his/(her) username into a file listing the allowed users.

In order to keep the root-password secure, it isn't stored in the application (or asked for) by the GUI. Instead a small special-purpose SUID-root'ed[5] commandline program (lpfwRootExec) have been made - capable of executing one instance of each necessary command (iptables-save, iptables-restore, lsof, netstat, lpfwIptablesQueueHandler, . . .).

The program-executions are all built over the same model: They obtain root-rights first with seteuid(). Secondly, they check the user's rights to execute. And finally - if the user is allowed - uses execv() to execute the command.

Before the actual execv()-call, they have checked that it is an allowed user who are executing them. That can be done by using getui() and getpwuid() to get the username of the 'real user-id' of the process. The Linux kernel always maintains to userid's for an process: real-uid (UID) and effective-uid (EUID). The real-UID are the user-id of the calling process, the effective-UID is the user-id of the executed program-file[6].

An example: if a program is set SUID-*root*'ed and run by a user logged in as *foobar*, the real-UID will be *foobar*, and the effective-UID will be *root*. Thereby, the program can see if obtained

[4]'Hey, that was fast! - the code just got written. . . '

[5]I.e. the program-file is owned by root and SUID-bit is set. (Most programs in the system-installation-folders are owned by root, but only few have the SUID-bit is set.)

[6]The getuid()-man page explains this in detail.

root-rights, and if the user executing the programme is in "the allowed list". The list is just a file listing the usernames of those allowed to run the root-enabled parts of our application. Of course, that file must be writable to root only!.

The exec() function is used to run the command. Before exec(), the stdin-filehandle could be closed for further security. This way, there no possibility to feed any input to the shell, once it's exec'ed, since exec() inherits the stdin, stdout and stderr-handles from it's mother-process. The mother process is not a parent-process, because the exec() takes over the process[7] and overwrites all program- and data-segments of the calling-process, so the mother is simply 'taken over' by the exec-call.

Several of the small programs could exists independently, and thereby improve the compile- and release-dependencies. But they have been collapsed into one console-application, since they don't change much, and basically they would all do the same thing: check the root-rights of the user an then execute a predefined command.

Debugging and testing root-console-code Care should be taken when running as root. The testing is thorough and the code is full of sanity-checks. Testing can be automated, but in this case it was done manually (by hand), by simply using the commandline. Good programming practice is to perform *only what is* <u>*strictly*</u> *necessary* as root (or kernel-module), and then - as soon as possible - revert back to a process-space with fewer execution-rights - i.e. our main program and continue the processing there. And this practice cuts down on our amount of test-cases.

When encountering problems, debugging usually isn't available e.g. as when buffer-problems occur. Because it's part of a collection of concurrent processes, it isn't an option to halt the program with the debugger - as it is in a parallel execution with the GUI-application on one side and possibly the kernel's network-stack on the other (the case for the LPFWIPTABLESQUEUEHANDLER).

So - as with the GUI-DragN'Drop-code - the only option is the old fashioned debugging approach: printf()'s - and loads of it. Since we are a independent console-application, the only channel available is stderr, which the GUI eventually posts on the log-window, catering for our debugging needs. By printing all sorts of internal data on to stderr, we get the insight of what is going on. Several levels of verbosity are available (Level 0-3).

7.7 Capturing packets (QUEUE-handler)

To capture packets, netfilter offers a special target: The QUEUE-target. It's a so called *netlink* to the kernel's network-layer, the new netlink-interface is to be used instead of the usual linux-kernel ioctl()-calls to control network-parts in the kernel. The netlink is termed a *berkley raw-socket*, which means it is a regular socket - as known by application-programs - but it's packets have bypassed most of the IP- and TCP-layers of the kernel, and are delivered as raw ethernet-frames, just as if it came directly from the ethernet card (the datalink-layer). That closely resembles the same interface as with *pcap*, i.e. when capturing packet using the pcap-library as *tcpdump*, *ethereal*, *snort* et al. uses.

The main difference between pcap and netlink is: The pcap-library puts the ethernet-NIC in promiscourus-mode (needs root-access) and recieves *all* packets on the wire. With the netlink, *only the packets we are intereseted in* - comes out of the socket, because the firewall-rules have filtered out the specific packets.

Packet data Since the netlink sends us raw ethernet-frames, it means that the the IP-header (and higher level headers) have not been stripped off yet, and the ethernet-data is still available (i.e. MAC-address etc.). The full payload of the packet can be extracted by firstly, casting the data to an ip-header-structure, and then (depending on the type of packet) to e.g. TCP- or ICMP-headers. Once the headers have been identified, the length-fields in the headers will reveal where the remaing payload-parts of the packet is.

[7]The exec()-man page explains this in detail.

Deployment and build-issues The interface to the QUEUE-target is though the *ipq*-library, which we have included in our source for build-convinience. But the ipq-lib is actually from the netfilter-package. The library has more simple and convenient functions to use, than plain raw berkley-socket-programming.

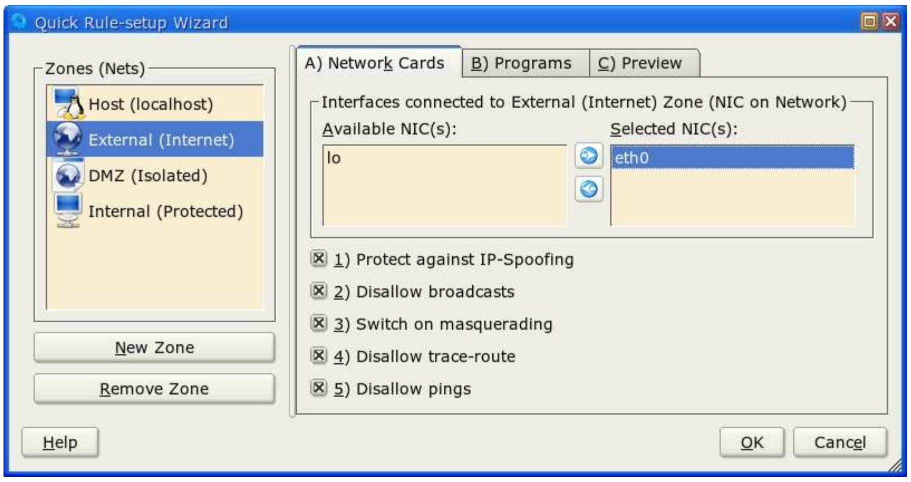
The QUEUE-target is present in standard kernels, so no special patching and re-compiling of the kernel needed by the user. But using the QUEUE-target comes with some thorns: 1) There can be only one process attached to the queue, so it's either e.g. *snort* or us - not both. 2) The packets are indistinguishable as to which rule captured them. Only that it was a rule with the QUEUE as target can be known. There is the possibility to use mangle-rule to set a mark to distinguish between packet from different rules, but that moves filtering rules from the filter-chains to the mangle-chains, since the marks can only be set in mangle-chains. It makes the rules unclear and messy.

Running and program-usage The demon runs very ordinaryly on the commandline, but it must run with root-rights, so its usually started by the LPFWROOTEXEC-utility (as with all the other root-commands). Its a plain and simple commandline-program in straight C++, that holds lists of unwrapped packets, holding housing both header and payload of the packets. It gets commands from the stdin, and writes to stdout, when new packets arrives. The currently held packets on the queue can be listed on stdout, and verdicts on packets can issued by typing 'A[packet-id]' on stdin (for ACCEPT). There is a complete usage-print available with option '-h'.

Since it's very independant, it can be used by *any* application wanting to catch specific packets using firewall-rules to pick out the packets. The packets can be investigated and possibly altered before handled back to the network-stack. I.e. it's a small independant utility.

7.8 Setup Wizard

The SetupWizard's purpose is get the user started with some sensible rules. It can be seen in Fig. 7.4, and have two main tasks to resolve: Which zones are attached to this host (which zones exists), and what programs (ports) should be open to each zone.



Interfaces connected to External (Internet) Zone (NIC on Network)
Available NIC(s): lo
Selected NIC(s): eth0

[✓] 1) Protect against IP-Spoofing
[✓] 2) Disallow broadcasts
[✓] 3) Switch on masquerading
[✓] 4) Disallow trace-route
[✓] 5) Disallow pings

Figure 7.4: Setup Wizard (in process of assigning NIC's to Zones. . .)

The wizard have two tabs for each zone, which allows the user to:

- Assign NIC's to network-zones, and thereby get the information of how the network is laid out (A-tab).

- Open up for some (well-known) programs, e.g. SSH, FTP, SAMBA, NFS etc. on each zone (B-tab).

The third tab was intended to show a preview of the rules that would be generated by the current choices, but it's less important, since there are many stops, where the rules can be seen and viewed, before the rules actually gets inserted into the firewall (e.g. the RuleView).

Additionally, there are some tic-marks that can be activated, which should append some handy and common rules - like checks for IP-spoofing, broadcast-spoofing, ICMP-responses (*ping* and *traceroute*) and NAT'ing (masquerading).

In functionality, the wizard isn't that different from the bash-setup scripts of FireHOL and SuSEfirewall - it will basically do the same as them. However it is a more graphical and user-friendly dialog than editing some text-config-file. That is why it should exist, not because it makes a better set of rules - it just makes them in a nicer way.

Rules The wizard will make rules by using the DCOP-interface on the RuleView. By inserting rules into the RuleView (under a new configuration), they can be checked and altered before deployment. The construction is intended to be more or less similar to the SuSEfirewall-script, and an example can be seen in Appendix A on page 93.

The FireHOL (See table 5.1) make some similar rules, though with less sanity-rules to check for anti-spoofing, -broadcasting etc. But, in contrast to SuSE, it have the concepts more in place. For instance, it opens up a *service*, e.g. SAMBA, which actually means it'll make rules that opens up several tcp-ports (137-139 and 445) - and not just specific numbers like SuSE (although you *can* specify that too). We would like to use an identical concept on the program-tab-page (B) as FireHOL.

The wizard got hit by poor API-design - as described in our strategy-section (See Sec. 3.3 on page 28), that could happen - and it showed up early during actual use. The wizard got started *very* late and didn't get finished, mainly because of lack of time.

The specific problem (apart from the late startup) was an less-optimal DCOP-interface on the RuleView, which made it very tedious and impractical to make large quantities of rules. A few rules by hand worked OK, but when inserting many becomes troublesome, because the DCOP-interface needs a handle to indicate which item to manipulate. We use the *sortkey* (See Sec. 7.5.3), but that was less fortunate, since we are auto-numbering the items to keep the sorting in order, so this handle changes every time we insert or delete some item - as the wizard would do. It can work, but every time the insertion-point must be found by using the findNode()-function to find the item in the view, before some edit can take place at that item.

It does work in practice, but it is necessary to re-design the DCOP-interface[8] - and most likely the sorting-issue of the RuleView as they seem coupled together. No doubt the listview-sorting sucks, but it is partly linked to the internals of the RuleView's listview - and it really is less unfortunate, that it has also been reflected in the view's DCOP-interface.

7.9 Development- and Test-environment

The tools used in our develoment-frame should be readyly available in any recent distribution. Like with the API-development (See 3.3 on page 28), we will be using external interfaces and tools in our own development-cycle to see if they are easy to use.

Tools Our tools are:

- Use of kdevelop, as editor and autoconf/automake-environment.
- Postgres-tools *psql* (SQL-client commandline-shell) and *pgaccess* (GUI-SQL-client) to manipulate and inject test-data into the SQL-database.

[8]Otherwise this programmer *will* go nuts while inserting the amount rules needed by the wizard...

- SuSEfirewall2-script to get some rules up quickly - and until the SetupWizard get working. RedHat have something similar, otherwise use FireHOL or some home-brewed scripting.
- CVS as revision control, primarily because sourceforge.net is also using it.
- Various standard command tools (like diff, grep etc.)

Testing The diversity and amount of code is large. The project's source-code[9] is 20.000[10] lines of handcrafted code, with an additional 8.000 lines being auto-generated by KDE's kconfig_compiler (config-settings-dialog-singletons), Qt's moc-compiler (signal- and slot-mechanism), and the XML-GUI-compilers (widgets, dialogs and windows). The project have come to a size, where some automated script-testing ought to be used. However, time didn't permit that, but its on the todo-list of things needed. So far, testing have been done ad-hoc.

Testing the code is actually a two-folded issue: A) The GUI needs testing and, B) The generated firewall-rules and parsers needs testing.

Testing GUI is notoriesly difficult to automate, so manually testing though trail-and-error will prevail, we'll try and use the application as much as possible for all our purposes.

The generated rules and algorithms have been tested by using a manual form of regression-testing: Starting from a known database-state (cleared/re-initialised or loaded a saved configuration/database), the same rules where parsed in, and they should generate the saved reference as previously approved as an OK reference-result. Additionally, the Postgres-tools have been used to load and save comma-separated (CSV-files) dumps to the database, and then diff the result of parsing a known configuration into the database.

Testing the commandline-programs have been simple: use a bash-shell to get them right, and then in the GUI, dump everything sent and recived to the sub-shells into the log-window. That have been quite sufficient to get the chatting though the std-in/-out/-err-channels right.

Additionally, the netfilter-package have some test-tools that allow some virtual ethernet-interfaces to be created, and to create and recieve packets using bash-scripts, e.g. for sending ICMP-reply-packets in test-scripts. More information on that can be found on [netfilter, Hacking Howto]. Those netfilter-test-tools enables automated regression-testing of the firewall-rules. However, we havn't actually used any of them yet, because so far, simple telnet- and SSH-tests have been sufficient.

[9]Counted with a "*cat $(find . -iname '*.cpp' -or -iname '*.moc' -or -iname '*.h' -or -iname '*.ui' -or -iname 'Makefile.am') | wc*" in a bash-shell

[10]For comparison: My former employment involved the development of a 600.000-lines application, which took 10 men two years to develop. That is roughly 30.000 lines per man/year - though, that code was a lot more well-designed, optimised, tested and documented, which cuts down the code-size to bare essentials.

Chapter 8

Concluding Remarks and Future work

The project revealed some specific opportunities and problems along the way, and here we present both: Problems first - then follows suggestions and opportunities.

For more general remarks on the development-method, please see the conclusion in Sec. 9.

8.1 Rule verification and integrity

To ensure the user gets the rules needed is the primary focus. In this project, no functionality have been made that seeks to verify and check the integrity of firewall-rules - generated as well as user-manipulated. It was recognised early in the project, *that this area is defficult*, and that the related academic work (See 5.3 on page 43) *is* very relevant for our project-goal.

However, it was decided to stick to the original focus point of providing a pragmatic workable solution now, and not an academic investigation into automatic firewall-configuration and verification. It was esimated (by me), that the author (myself) doesn't have the interlectual capacity to think up a shiny new solution, that does any better than any of the suggestions and prototypes already out there on the Internet.

However, the design have been made, so that such verification-tools could be fitted into the frame-work, and that is as close as this project gets at formal verification and integrity-checking the rules.

But, some thoughts have been contemplated about checking the rules for integrity, idioms and contradictions. Firstly, one could differentiate between:

Pragmatic integrity Using common sence - e.g. by deleting the rule that allows all loopback-traffic, the X-session is killed - which the user is using to run our X-Window program in. It's not illegal in a mathematical sense - it's just stupid.

Formal integrity Using mathematical means to ensure that e.g. a rule isn't later being contradicted - or that un-reachable rules don't occur.

The pragmatic approach is of a practical nature, and relates very much to experience and environment - i.e. what ordinary users would do and the context with other tools used on the platform (e.g. like the XWindow-'tool'). It's plain programmer-grease just like GUI-UI and setup-problem-solving.

The formal approach is what *fang*, *firmato* etc. are focusing on. They mostly rely on a logic-specification of the security policy. But in order to make a logic-specification, it s necessary to 'know' what a rule means - i.e. when does it activate (match) and what does it do (target/verdict).

That is possible with ACCEPT/DENY-rules with predefined matches, but what about 'let user-space-program XYZ decide' - how does the logic handle that scenario?

Some examples to illustrate the point:

Example 1 In a dedicated input-chain for IP's in range 10.0.0.X is a rule that filters on destination ip 192.168.100.Y - clearly that rule will never be matched, because only packets to 10.0.0.X will end up traversing the chain anyway - and a rule checker could find such anomaly.

Example 2 A deny-rule of 'match-packets-for-strings-with"#!/bin/sh"-on-port-22' (i.e. an extension module), followed by an general accept-rule of any traffic on port 22 - How should the logic handle that?
It doesn't know the special extension module and might have trouble making sense of the deny-before-accept-sequence, which is generally considered unhealthy practice (commonly, it's accept-special, deny-in-general).

Context and organisation of rules also plays a role.

In example 1, the input-chains where organised into IP-address-ranges, and further organisation into such trees of chains (table -> nic -> ip -> port(s)/service) is very common (e.g. see the SuSE-config in Appendix A). The logic will (must) take this into the evaluation, and it makes it problematic to re-arrange the rules. E.g. to deploy a trick like in example 2, where we do a drop-wierd-packet-before-accept-normal-packet.

Firstly, I would think that, it implies the logic will have to define what constitutes a normal packet. And secondly, it forces the rule's placement in the tree, to become part of the knowledge of what the particular rule means - meaning all rules leading to this part of tree, must have been known and identified to get the full match and meaning of the particular rule.

Some of the verifyers using CLP[1], do this, but such need the full knowledge leading up to every rule, in order to make the logic-statements, and some also show how to descripe a packet as being normal - using CLP.

...but som reason, I can't shake off this impression: That it doesn't look fool-proof to me.

Admitted - I know very little about logic-programming, but I don't see this CLP-approach as mature enough to capture all the needed firewall-capabilities in daily use. And as long that the solutions also require Logic Programming skills, I don't see them as ordinary solution for ordinary users (See also the Summary in Sec. 5.3 on page 43).

The CLP-approach at least, looks at the rules with a closed world assumption, it knows all possible meanings of rules - and that doesn't play well with 3.party extension ideas (unless they too are specified in CLP).

An alternative is the OO-suggestion (sketched in Sec. 8.2), where we try to focus on the parts we *do* know and allow us to ignore - or be unaware about the rest. But that implies a language (and a compiler) to specify the various parts and sections. The language must provide scope, identity and extension, mostly matched by the encapsulation and polymorphism of OO-languages like C++.

The problem still persists However, this project also gets hit by the same trouble as every OpenSource-project in Sec. 5.2 on page 38, and that occupy the work in Sec. 5.3 on page 43: To find stuff, you needs to know where to look, and what you are looking at, in order to indentify and find e.g an insertion-point for a rule... which is pretty much the same problem that verifier needs to solve too.

It was also contemplated whether or not we needed a language (i.e. a compiler) to express sections and iptables-rules - e.g. for identifying the section where the SetupWizard would open up for a service.

[1]Constrained Logic Programming - e.g. using Prolog or Eclipse (a knowledge-database with predicate logic-statements run through an inference engine).

The total network-graph-view Our grand idea of showing an editable graph like Fig. 6.5, using the dot-tool, have also been halted by the issues above of recognising the rules.

It was thought that our dot-graph-view would show packet-paths, as dictated by the rules. The paths/archs/lines are matches: I.e. rules and routing between tables, - and the nodes/boxes are the targets: Tables (used by routing having target-decisions-defaults), chains (custom-target, but undecided) and target-decisions (accept/drop/mark etc.). Note, the graph doesn't show sequence of the rules, it shows access-paths. The sequence is shown implicitly by only drawing accesible arcs, and that is controlled by the 'graph-order'-algorithm.

The sequence of rules leading to targets are collapsed in the graph, by intelligent interpretation of the rules. E.g. within a sequence of specific 'accept-tcp-port'-rules, there can be one single general drop-all-rule, which then renders the remaining rules use-less - and hence, these rules must not be drawn, because they are never reached! And that is an application of a rule-checker/-verify'er.

So, in order to draw the graph, you need to know the rules: i.e. what each of them does. And if you know what a rule does, you propably can say something about if it a good idea - and doesn't that mimic the work of a rule-checker/-verifier?

And, how do you draw things you don't know about? E.g. rules with ip-addresses, ports and (known) targets of DROP/ACCEPT - are drawable. But what about a rate-limiter? Is it split-ended with one arrow (e.g. from INPUT-chain-icon) to process-space and another arrow to DROP (e.g. Trashcan-icon), because some packets make it - others don't?.

It can be questioned - that if not *all* rules can be drawn - what's the value of the rules that are drawn? I.e. one can *almost* see the firewall-rules in the graph... - execpt the once that can't be drawn. Of course, a compromise can reached, by at least showing in the RuleView what could/couldn't be drawn.

But the value of the graph is still up for debate, and that one of the resons why it ain't there (yet;-).

8.2 A concept of Object-Oriented firewall-configuration

Perhaps it's possible to take on a different approach to the problem of recognising rules, at least with the purpose in mind to manipulate them. It is a compromise-suggestion that segments rules into recognisable sections, where a section then can be laid out in rigid, strict and well-known patterns. That allows both flexibility and diversity, along with a stringent layout in particular sections, as a module can then find its section and deal with only within it's own section.

The amount and variation in rule-parameters and -targets, organisation of rules and the network layouts, makes configurations diverse and complex. So when things grow out of hand, applying a *divide-and-conqueror*-strategy usually helps - and the object-oriented approach is such.

It is possible to approach the configuration of firewalls in a procedural and object-oriented way: think of configurations as classes with inheritance-capability. The tables and chains would resemble variables (holder of data and organisation/structure) and rules would resemble functions (performers/actions).

The three paradigms of OO is *Encapsulation*, *Inheritance* and *Poly-morphism*. It is possible with achieve some properties of these three issues:

Encapsulation Each configuration, table and chain holds lists and a such provide a grouping of type and contents. Little enforcement is present though: there are no 'private' parts in the rules.

Inheritance A configuration could inherit another: Tables, Chains and Rules not existing in the base, are appended by the derivative to form a combined configuration.
Tables are hardwired into the kernel and are merge-able by appending the chains of the tables. Chains are appended when they differ in name and merged when they are similar.
The merging can provide conflicts when the rules are differing in contents, but if the derived chain always append it's rules - and rules never are identical with some in it's base - it should

work in a similar fashion as with C++-classes.
Their derived classes's memory-allocation of the class-structure and the V-table-layout (for virtual functions: function-pointers) are appended to the base by it's imidiate derived class. But declarations within class- (or in our case: config-) scope must be unique and no-overlapping.

Polymorphism A rule must match some criteria and point to a chain as a target (jump) before the chain can ever dream of being traversed (except for the builtin-chains: INPUT, OUTPUT and FORWARD. They are the default starting points of traversal - the kernel jump to these targets unconditionally when starting traversal of a packet). By letting the rule stay in the base-configuration, and the chain in some derived configuration, it becomes similar to a virtual function in C++. The rule resembles the pure virtual function definition and the chain the derived polymorph-ed implementation.

It will require some form of standardised way of setting up rule-sets, that allow a main-skeleton to be recognised - i.e. to find the main-hooks. Then, chains can be inserted, for wich a set of operations are confined within. That way, the section (chain) can contain only well-known rules, and the layout within the chain can be as free as needed. Also, the main standardised structure allows modules to find injections-points when creating these sections.

This was thought of fairly late in the project, but it would be a way to model the setup and avoiding the fundamental problem: If insertion or deletion of parts of the configuration is to be done - then it must be possible to find the insertion and deletion-points.

That implies, searching though the configuration for the right spot, which can be done in two ways: 1) look for a marker, or 2) figure out the logical structure to find the 'correct' spot.

The second method is the most correct one, but it implies that all possible meanings of rules and the organisation of them must be recognisable and well-known in function. That is difficult for simple cases of firewalls and next to impossible with an open, extendible and flexible firewall like netfilter/iptables.

The first method means the marker must be unique, present and in an expected, identifiable layout (with the operative area of the insertion-point, e.g. within the searched chain). But that is far more achievable, because only a fraction - the well-known operative area - are dealt with. All other rules and chain-layouts are ignored and remains unknown.

The object oriented approach supports this thinking: An algorithmic functionality/solution should only deal within a particular chain (and with rules), which is known and it fully understand in layout and contents. Encapsulation is the issue here and should support that. Activation is unknown by the algorithm, but polymorphism takes care of it. The combined functionality of algorithms and manually tuned parts are achieved by Inheritance - letting rules calling chains as targets.

Admitted, its a loose suggestion, but deploying programming-languages on OS-platforms are the inspirational source. E.g. the C-link-point for the main()-function is such a recognisable point, with C++-virtual-function-tables as the other inspirational point. The diversity in programming and possible layout of code, can be combined with well-known and recognisable layouts and hooks in that world. Perhaps the same model will work for firewall-rules?

8.3 Future Modules for lpfw

During the project-period several ideas and oppertunities have appeared for modules to include in this project - the 'Linux Personal Firewall' (lpfw). Here are some of the more persistant ideas that keept popping up in my head.

8.3.1 Setup-Wizard GUI-Frontends

With GUI-parts for E.g. the SuSEfirewall2-script, FireHOL and possibly others, other people's effort could be provided and used with in the application-frame. Most of the frontends would be wrappings without a great deal of extra functionality, but for instance, the ProcView's knowledge of network-layout could be used (though DCOP) to fill in the NIC-definitions in E.g. the SuSEfirewall2- or FireHOL-config-files (See also Appendix A for SuSE and table 5.1 for FireHOL).

8.3.2 Rule-checkers and -verifyers

The work of *fang*, *firmato* and *ITVal* could be wrapped up in modules. It should be quite possible, especially for fang, as it already written with a Qt-GUI. The ITVal is aimed directly at iptables, which makes it more relevant and less work than using fang. It is commandline-oriented, so it too seems fairly easy to fit in a module.

More notes in Sec 5.3 on page 43.

8.3.3 Routing parser and NetworkView

The routing and network-interfaces should be dissected and presented, preferably as a dot-graph. Currently, the information is used only in the startup-wizard, and the module 'GUI: NIC view', shown on the system design schematic (Fig. 6.9 on page 61), isn't implemented as an independent module. Instead, it is housed within the ProcView-module for now, due to shortage of time.

The SetupWizard only deals with opening ports on the firewall-host, so it'll only insert rules into INPUT- and OUTPUT-chains (See Sec. 5.1 on page 35). There is no handling of FORWARDing, and this network-view is the place to specify such, since its a matter between hosts in zones.

It would establish the 'third leg' in the tripod of a) processes, b) firewall-rules and c) NIC-network-layout. What's to be in it haven't been established clearly yet, but it is a missing piece, because the SetupWizard get you started once, but a NicView will provide a place to put functionally for reacheable machines on the internal network (e.g. using *nmap*), re-allocation of NIC's to network-zones, and in general provide the next step: from this firewall-host to the immidiate network.

8.3.4 Statistics and quota

In order to achieve better logging, the ulogd-demon[2] could be used. It is capable of more versatile logging than the plain LOG-target, and its also in the standard netfilter-package. It sends all logs to a SQL-database (e.g. to Postgres), and all it takes is to change LOG-rules to ULOG-rules. So, apart from setting up and running the ulogd-demon on the host, some very simple rule-replacements are needed.

The better and more structured logging allows better filtering using SQL, which is fairly straight forward to present in the DBView - or in a view by it self. With statistics, it becomes possible to single out and measure specific traffic, for e.g. throttling by temporarily holding packets on the queue, and then adjusting the time the packets are on held on the queue. It's a alternative SQL-based quota-solution made in user-space, which could be much more flexible and doesn't require fiddling with such things as 'token-buckets' in the traffic-control paradigm.

8.3.5 Process-signature checking

By knowing what process is involved in the communication with a particular captured packet, the possibility arise to check the binary signature of that process. And e.g. verify that it really *is* ftp-client X, in version Y, with latest patch Z, which is the originator of the packet going out - and not some P2P-client.

[2]See also ulogd 1.21 - User-space Packet Logging for netfilter at http://gnumonks.org/projects/

And noteworthy: that check is done without knowing about or checking the packet-content (as e.g. *snort* can/will.), it check against a list of allowed program-binaries - e.g. by querying a host-integrity-application like *tripwire* or *AIDE*, which holds the current list of checksums of allowed program-binaries.

8.3.6 Network-wide approach

All that can be done with the program on the this - the local host, can also easily be done on a remote Linux-box. Our program uses command-lines (See Fig. 6.9 on page 61) to get/set data on the host. It simply opens up a local shell and issues the commands. By using SSH as the shell instead, the exact same information can be extracted or set on any host - i.e. the remote host to which we have a SSH-connection to.

The system already have the design-structure to handle it: the database can easily handle multiple configurations, and it don't care if its on this host or some remote one. The tools, which gets/sets data in the OS at the various OSI-layer-levels, are all command-line-tools (NETSTAT, PS, LSOF, LPFWQUEUEHANDLERD, LS, etc...) - and they are handled though an opened shell: e.g. a plain bash- or SSH-shell. All it takes, are the installment of the command-line-tools and an active SSH-account on the remote host. The SSH-account don't even have to a root-shell - just list the SSH-account's username in the allowed-users-list on that host, and *only* the command-line-tools in question are executable by the account.

This takes the program to to network-wide level, where it becomes possible to:

A) Manage multiple iptables-firewalls from this host, extending this program into a system-manager tool (although the '*personal*'-term should probably be dropped from the application-name...).

B) Still on this firewall: To extend the packet-capture-process from just the INPUT/OUTPUT-chains, to that of the FORWARD-chain too. Because now, it can check the end-socket-data on those hosts within the internal network too - that is: The process on *that* host, which sent/gets the packet being captured in *this* firewall's forwarding-chain. It is easily achieved too, just substitute the opened bash-shell with that of an SSH-shell to the remote host instead. But it haven't been pursued in every detail, due to time constrains.

This is a very desirable feature, since it also opens up for the possibility to check the binary signature of that process on that remote host. And e.g. verify that really is ftp-client X, in version Y, with latest patch Z, which is the originator of the packet going out - and not some P2P-client.

Our design-structure have been built with the above features in mind, thereby capturing some of the network-wide features of e.g. *Firewall Builder.* (See Sec. 5.2 on page 38).

8.4 Future Modules for netfilter

Some extensions to netfilter could be useful, though there already are a lot of extensions[3]. A basic problem is that with most extensions, the kernel needs to be patched (using patch-O-matic) and re-compiled/-installed, which is out of reach for most average users.

But especially for debugging, tracing and logging the route taken by a packet through the firewall, some alternate solutions could be provided with the following extensions:

- Tee-target: Wraps a another target. It will execute the wrapped target as normal *and* tee off a copy to another target.
  It could be used along these lines:
  "iptable -m .... -t TEE stats | ACCEPT" or "iptable -m .... -t TEE $(echo 'ruleXYZ matched' | syslog) | DROP"
  That whould collapse the build-in non-terminating Log-target with the next rule for which

[3]See them at http://www.netfilter.org/patch-o-matic/ under the various repository-links.

it is logging. E.g. today, a rule-pair for looging and action on port 21 (FTP) would look like this:

```
pkts bytes target prot opt in out source destination

548 278 LOG tcp -- eth0 * 0.0.0.0/0 172.16.50.59 tcp dpt:21 flags:0x16/0x02 LOG flags 6 level 4 prefix
'SFW2-FWDdmz-ACC-REVMASQ '

548 278 ACCEPT tcp -- eth0 * 0.0.0.0/0 172.16.50.59 state NEW,RELATED,ESTABLISHED tcp dpt:21
```

Basically, such logging would (...does!) double the amount of lines in the firewall, and it becomes un-easy to see whats being logged and to distinguish pairs.

- Delay-target: Simply delays packets by holding them for a while - for slowing down a matching and thereby a connection
- Quota-target: Like the built-in target 'Limit', just limiting bandwidth-speed instead of connection-attempt. It resembles the Delay-target, but most likely rejects the packet and delays the rejection to achieve the slowdown.
- NOP-Wrap-target: Wraps a another target but don't execute the target - (**No**-**OP**eration-wrapper) good for testing, debugging or temporarily disabling a rule.

User-space execution of choices regarding matching and targets are needed. The built-in Queue-target is such a solution, but it seems that there can be only one Queue, and all packets are delivered on one and the same virtual NIC-interface.

It is desireable to have diffent packets on different Queues, e.g. one for each rule. Since the firwall already have sorted the packets by various matching criteria, why collapse all packets into one stream again ?

- MultipleQueue-target: For having several user-space programs running at the same time - i.e. several Queues.
  Some attempts exists like *ipqmpd* 0.3 - IP Queue Multiplex Daemon[4], but it was a bit unclear to me *exactly* how it works, and it doesn't seem to have been updated in years.
- ExecCmd-match and/or -target: This would execute a command that possibly decided a possible match or a particular result (target) of the packet. This is possibly more convinient to used the DCOP-interface in the ProcView, since it can be made to get access to the captured packets and their payload - for now, it only looks at the headers.
  Such could be used to perform a command that could query an host-integrity-service, and verify the integrity of the program-file of the process, that is involved in the transmission of the packet. This way, the firewall-rule can ask the host-integrity-service if the executable is ok, and not an virus-infected-version, a trojan horse-version or a non-updated pathed version of a program (e.g. is it really "FireFox Ver 1.0.1a" that is trying to send a packet ?).

The netfilter-extensions are running in kernel-space, so the amount of advanced fiddleing with packets and related them to other data is to be keept to a minimum. If it is possible at all, the extensions should be user-space programs instead, making it more safe and free us for *patch-O-matic* and re-compiling the kernel.

8.5 Summary

The DBView, RuleView and ProcView could be used to get access to databases, rules, processes and packets - and a lot of the suggestions could be made in user-space instead - as a KPart/KPlugin etc.

That is one of our foremost goals of this project: To aid and create a frame for levering the development and deployment of others. It is hoped that the solutions and extensions sketched out in this chapter shows that purpose, and that plenty of opportunities have been made more readily available.

[4]at http://gnumonks.org/projects/

Chapter 9

Conclusion

The project succeeded in achieving the most of the core requirements. The goal (Sec. 2.4) was to create a modularised framework for firewall-configuration - and subsequently, to partly or fully automate setup and daily use of the firewall. Of the requirements listed in Sec. 4, the A1-to-A4, B1, partly B2, B3-to-B7 and C1 - have been accomplished to a satisfactory extent.

Note, that B2 (forwarding), B7 (new rules) and C2-to-C6 have not (or to a lesser degree) been meet. The C2-to-C6 where intended as futuristic opportunities that could be feasible to make, if time permitted. Therefore, the priority of C2-to-C6 was DESIRABLE (see Sec. 3.2.1), B7 as just plain hard to make, and hence DESIRABLE.

The framework is flexible, scales well, and is an independently built and released entity. Its exclusively having platform-requirements, which are readily available on any recent standard Linux-distribution. Though, the installation procedure still needs some work, before average users can install it.

The framework have a multi-layered interface, both API- and UI-wise. It allows users to operate at any level of detail - as they feel comfortable with and/or needy of. Programs can hook into any layer and module, and enhance or extend functionality through interfaces, using KParts/KPlugins, DCOP or SQL. It might actually aspire to become the first available C/C++-API for iptables.

The daily operation of the program achieved some success. Most fundamentally, the link between rule, network-stack, packet and process was established in the ProcView, enabling the Windows-like *circuit level-packet-filter* to work, by halting packets and allow the user to dynamically open up per connection-attempt. The RuleView enables round-trip-maintenance of both constructing and deploying rules - and reading them back in again from the Linux-kernel's netfilter. Although the RuleView is a bit too detailed for averages user, expert-users should be satisfied.

The overall idea of *trust, but verify...* was achieved by verbosely showing, whatever the program is doing, and allowing the user to approve any changes - at all layers and levels. The RuleView, ProcView - and in part the DBView - serves this purpose.

The resulting framework-structure and the UI-layout for daily operations, succeeded in achieving the overall wish of an open and transparent program. It is still operating entirely on the individual host, but can easily be extended to becomes network-wide as described in Sec. 4 and 8.3. The forwarding isn't in place yet (requirement B2), and it most likely will need a routing/network-view like outlined in Sec. 8.3.3, before handling of forwarding-rules becomes presentable.

The automation-work have some work left in areas like the SetupWizard and the installment-procedure, before average users can operate it. The installment requires fiddling with installment of both the Postgres-database, the hardware-setup of NICs and routing, and the creation of some rules - before the program can be used with satisfaction.

The problem of finding and manipulating rules - while still retaining an open interface to others

- haven't been resolved. The closest workable solution is something along the lines of cooperation - like the OO-idea. Searching in, and merging diverse sets of rules, requires either one module with full knowledge of all possible rules and layouts. Or it requires cooperation among several modules, by adherence to a common standard. The SetupWizard is intended to use this idea. Until then, use of already available setup-tools like SuSEfirewall2 or FireHOL is advised.

Our strategy (Sec. 3) paid off in some areas, like in try using our own interfaces (Sec. 3.3) and in the iterative, Darwinistic approach of what works easily - and what doesn't. But some frustrations did arise when using components, like e.g. KDE. Overall, the iterative and sectioned approach in our strategy worked well in handling the diversity in skills, requirements and coding-diciplines.

Generally speaking, iterative agile component-based development *does* pays off - it's good, but isn't as fast-paced as expected. It doesn't cut development time - instead it heightens the amount of features, the production quality and results presenting a uniform application.

For instance, with components - development-time is more spend on finding out *'how-to-**use**-a-component-that-does-database-editing'* (black-/white-boxing), than that of *'how-to-**make**-a-component-that-does-database-editing'* (homegrown code). Results are, that time-savings are partially postponed until a future reuse-scenario.

Development-implementation and -tests are still present in the same amounts as before, but more focused on functionality and features instead of spending it on infrastructure for e.g. GUI-layout.

Components allows and indulges several layers of interface. This allows better modularisation, enables future integration (more hooks and handles), and it makes testing much more readily available. E.g. by using pgaccess and diff with SQL for testing parser- and database-issues, using DCOP (RuleView/ProcView), commandlines (QueueModule) etc.

By using components, the application becomes nicely rounded in many aspects. Since to a great extend, the little things like consistency in icon-, menu-, toolbar-, window-handling makes the presentation more uniform.

Does our solution work? - Yes, as soon as bugs, quirks and setup-issues have been fixed.

That means, that overall usage for average users isn't quite there yet - but it will be soon. Framework-wise the structure works fine, and programmers and advanced users can use the solution.

The first alpha-release on sourceforge is pending these fixes, and when they have been completed - it'll be working great! - rocket-science in your pocket...

Appendix A

Firewall configuration example

Her an example of an iptables based firewall with three networkcards, catering for three network-segments: Internal (Net: 192.168.x.x), External (Net: 10.x.x.x) and DMZ (Net: 172.16.x.x). It is a dump of the rules generated with SuSEfirewall, which constitues an ordinary way of organising the rules.

It have the default-drop policy, where it specifically allows particular ports and drops anything not allowed. It is a diode-firewall, restricting the incomming and forwarding chains, but default-allows all outgoing traffic from the localhost.

There are some sanity-rules for anti-spoofing and -broadcasts, as well as e.g. dynamically discovery of the mountd-port for NFS-rules etc. All in all some nice features and with a proven and tested track-record.

Here is the SuSEfirewall2-config-file:

```
# Copyright (c) 2000-2002 SuSE GmbH Nuernberg, Germany.  All rights reserved.
# Copyright (c) 2003,2004 SuSE Linux AG Nuernberg, Germany.  All rights reserved.
#
# Author: Marc Heuse <feedback@suse.de>, 2002
#
# If you have problems getting this tool configures, please read this file
# carefuly and take also a look into
#  -> /usr/share/doc/packages/SuSEfirewall2/EXAMPLES !
#  -> /usr/share/doc/packages/SuSEfirewall2/FAQ !
#  -> /usr/share/doc/packages/SuSEfirewall2/SuSEfirewall2.conf.EXAMPLE !
#
# /etc/sysconfig/SuSEfirewall2
#
# for use with /sbin/SuSEfirewall2 version 3.1 which is for 2.4 kernels!
#
# ------------------------------------------------------------------------    #
# PLEASE NOTE THE FOLLOWING:
#
# Just by configuring these settings and using the SuSEfirewall2 you are
# not secure per se! There is *not* such a thing you install and hence you
# are safed from all (security) hazards.
#
# To ensure your security, you need also:
#
#   * Secure all services you are offering to untrusted networks (internet)
#     You can do this by using software which has been designed with
#     security in mind (like postfix, apop3d, ssh), setting these up without
#     misconfiguration and praying, that they have got really no holes.
#     SuSEcompartment can help in most circumstances to reduce the risk.
#   * Do not run untrusted software. (philosophical question, can you trust
#     SuSE or any other software distributor?)
#   * Harden your server(s) with the harden_suse package/script
#   * Recompile your kernel with the openwall-linux kernel patch
#     (former secure-linux patch, from Solar Designer) www.openwall.com
#   * Check the security of your server(s) regulary
#   * If you are using this server as a firewall/bastion host to the internet
#     for an internal network, try to run proxy services for everything and
#     disable routing on this machine.
#   * If you run DNS on the firewall: disable untrusted zone transfers and
#     either don't allow access to it from the internet or run it split-brained.
#
```

```
# Good luck!
#
# Yours,
#       SuSE Security Team
#
# ------------------------------------------------------------------------
#
# Configuration HELP:
#
# If you have got any problems configuring this file, take a look at
# /usr/share/doc/packages/SuSEfirewall2/EXAMPLES for an example.
#
#
# All types have to set enable SuSEfirewall2 in the runlevel editor
#
# If you are a end-user who is NOT connected to two networks (read: you have
# got a single user system and are using a dialup to the internet) you just
# have to configure (all other settings are OK): 2) and maybe 9).
#
# If this server is a firewall, which should act like a proxy (no direct
# routing between both networks), or you are an end-user connected to the
# internet and to an internal network, you have to setup your proxys and
# reconfigure (all other settings are OK): 2), 3), 9) and maybe 7), 11), 14)
#
# If this server is a firewall, and should do routing/masquerading between
# the untrusted and the trusted network, you have to reconfigure (all other
# settings are OK): 2), 3), 5), 6), 9), and maybe 7), 10), 11), 12), 13),
# 14), 20)
#
# If you want to run a DMZ in either of the above three standard setups, you
# just have to configure *additionally* 4), 9), 12), 13), 17), 19).
#
# If you know what you are doing, you may also change 8), 11), 15), 16)
# and the expert options 19), 20), 21), 22) and 23) at the far end, but you
# should NOT.
#
# If you use diald or ISDN autodialing, you might want to set 17).
#
# To get programs like traceroutes to your firewall to work is a bit tricky,
# you have to set the following options to "yes" : 11 (UDP only), 18 and 19.
#
# Please note that if you use service names, that they exist in /etc/services.
# There is no service "dns", it's called "domain"; email is called "smtp" etc.
#
# *Any* routing between interfaces except masquerading requires to set FW_ROUTE
# to "yes" and use FW_FORWARD or FW_ALLOW_CLASS_ROUTING !
#
# If you just want to do masquerading without filtering, ignore this script
# and run this line (exchange "ippp0" "ppp0" if you use a modem, not isdn):
#   iptables -A POSTROUTING -t nat -j MASQUERADE -o ippp0
#   echo 1 > /proc/sys/net/ipv4/ip_forward
# and additionally the following lines to get at least a minimum of security:
#   iptables -A INPUT -j DROP -m state --state NEW,INVALID -i ippp0
#   iptables -A FORWARD -j DROP -m state --state NEW,INVALID -i ippp0
# ------------------------------------------------------------------------
## Path:        Network/Firewall/SuSEfirewall2
## Description: SuSEfirewall2 configuration
## Type:        yesno
## Default:     no
## ServiceRestart: SuSEfirewall2_setup
#
# 1.)
# Should the Firewall run in quickmode?
#
# "Quickmode" means that only the interfaces pointing to external networks
# are secured, and no other. all interfaces not in the list of FW_DEV_EXT
# are allowed full network access! Additionally, masquerading is
# automatically activated for FW_MASQ_DEV devices. and last but not least:
# all incoming connection via external interfaces are REJECTED.
# You will only need to configure 2.) and FW_MASQ_DEV in 6.)
# Optionally, you may add entries to section 9a.)
#
# Choice: "yes" or "no", if not set defaults to "no"
#
FW_QUICKMODE="no"


## Type:        string
## Default:     auto
# 2.)
# Which is the interface that points to the internet/untrusted networks?
#
# Enter all the network devices here which are untrusted.
#
# Choice: any number of device names, separated by a space. The
# keyword "auto" means to use the device of the default route.
# e.g. "eth0", "ippp0 ippp1", "auto"
#
# Note: alias interfaces (like eth0:1) are ignored
#
FW_DEV_EXT="eth-id-a0:56:08:5b:b4:19"

## Type:        string
#
# 3.)
# Which is the interface that points to the internal network?
#
# Enter all the network devices here which are trusted.
# If you are not connected to a trusted network (e.g. you have just a
# dialup) leave this empty.
#
# Choice: leave empty or any number of devices, seperated by a space
# e.g. "tr0", "eth0 eth1 eth1" or ""
#
FW_DEV_INT="eth-id-04:10:35:84:75:dc"

## Type:        string
#
# 4.)
# Which is the interface that points to the dmz or dialup network?
#
# Enter all the network devices here which point to the dmz/dialups.
# A "dmz" is a special, seperated network, which is only connected to the
# firewall, and should be reachable from the internet to provide services,
# e.g. WWW, Mail, etc. and hence are at risk from attacks.
# See /usr/share/doc/packages/SuSEfirewall2/EXAMPLES for an example.
#
# Special note: You have to configure FW_FORWARD to define the services
# which should be available to the internet and set FW_ROUTE to yes.
#
# Choice: leave empty or any number of devices, seperated by a space
# e.g. "tr0", "eth0 eth1 eth1" or ""
#
FW_DEV_DMZ="eth-id-07:8b:a3:71:fa:43"

## Type:        yesno
## Default:     no
#
# 5.)
# Should routing between the internet, dmz and internal network be activated?
# REQUIRES: FW_DEV_INT or FW_DEV_DMZ
#
# You need only set this to yes, if you either want to masquerade internal
# machines or allow access to the dmz (or internal machines, but this is not
# a good idea). This option supersedes IP_FORWARD from
# /etc/sysconfig/network/options
#
# Setting this option one alone doesn't do anything. Either activate
# massquerading with FW_MASQUERADE below if you want to masquerade your
# internal network to the internet, or configure FW_FORWARD to define
# what is allowed to be forwarded!
#
# Choice: "yes" or "no", if not set defaults to "no"
#
FW_ROUTE="yes"

## Type:        yesno
## Default:     no
#
# 6.)
# Do you want to masquerade internal networks to the outside?
# REQUIRES: FW_DEV_INT or FW_DEV_DMZ, FW_ROUTE
#
# "Masquerading" means that all your internal machines which use services on
# the internet seem to come from your firewall.
# Please note that it is more secure to communicate via proxies to the
# internet than masquerading. This option is required for FW_MASQ_NETS and
# FW_FORWARD_MASQ.
#
# Choice: "yes" or "no", if not set defaults to "no"
#
FW_MASQUERADE="yes"

## Type:        string
#
# You must also define on which interface(s) to masquerade on. This is
# normally your external device(s) to the internet.
# Most users can leave the default below.
#
# e.g. "ippp0" or "$FW_DEV_EXT"
FW_MASQ_DEV="$FW_DEV_EXT"

## Type:        string
#
# Which internal computers/networks are allowed to access the internet
# directly (not via proxys on the firewall)?
# Only these networks will be allowed access and will be masqueraded!
#
# Choice: leave empty or any number of hosts/networks seperated by a space.
# Every host/network may get a list of allowed services, otherwise everything
# is allowed. A target network, protocol and service is appended by a comma to
# the host/network. e.g. "10.0.0.0/8" allows the whole 10.0.0.0 network with
# unrestricted access. "10.0.1.0/24,0/0,tcp,80 10.0.1.0/24,0/0tcp,21" allows
# the 10.0.1.0 network to use www/ftp to the internet.
# "10.0.1.0/24,tcp,1024:65535 10.0.2.0/24" is OK too.
# Set this variable to "0/0" to allow unrestricted access to the internet.
#
########FW_MASQ_NETS="0/0"
########################################################################
## allow internal net all services:
my_int_FW_MASQ_NETS="192.168.100.100/24"
############# OUTGOING CONNECTIONS ################################
##### SEE also FW_FORWARD_MASQ for incomming connections ##########
#################################################################
## allow only approved services from dmz:
## allow outgoing:
##        dns (domain/53),
##        ntp-time (123),
##        ssh(to 192.38.95.24 only),
##        edonkey(test wierd high-ports)
## ...traffic
### DO NOT put host-names in th FW-rules - only ip-numbers !!!
### DNS needs to resolve them, but the FW isn't open yet!!!
host_dns="212.242.40.3/24"
host_clock="84.243.240.2/32"
host_ssh="192.38.95.24/32"
my_dmz_FW_MASQ_NETS=\
" \
172.16.50.59/32,$host_dns,udp,53 \
172.16.50.59/32,$host_clock,udp,123 \
172.16.50.59/32,$host_ssh,tcp,ssh \
172.16.50.59/32,0/0 \
"
##... What ports does the edonkey use?
##  It can run over any port. The defaults it uses are:
##  TCP port 4661 to connect to the server (admin).
##  TCP port 4662 to connect to other clients.
##  UDP port 4665 to send messages to servers other than
##  the one you are connected to....

## allow internal net all services - but only approved services from dmz
FW_MASQ_NETS="$my_int_FW_MASQ_NETS $my_dmz_FW_MASQ_NETS"
###############################################################################
## Type:        yesno
## Default:     yes
#
# 7.)
# Do you want to protect the firewall from the internal network?
# REQUIRES: FW_DEV_INT
#
# If you set this to "yes", internal machines may only access services on
# the machine you explicitly allow. They will be also affected from the
# FW_AUTOPROTECT_SERVICES option.
# If you set this to "no", any user can connect (and attack) any service on
# the firewall.
#
# Choice: "yes" or "no", if not set defaults to "yes"
#
# "yes" is a good choice
FW_PROTECT_FROM_INTERNAL="yes"


## Type:        yesno
## Default:     yes
#
# 8.)
# Do you want to autoprotect all running network services on the firewall?
#
# If set to "yes", all network access to services TCP and UDP on this machine
# will be prevented (except to those which you explicitly allow, see below:
# FW_SERVICES_{EXT,DMZ,INT}_{TCP,UDP})
#
# Choice: "yes" or "no", if not set defaults to "yes"
#
FW_AUTOPROTECT_SERVICES="yes"


## Type:        string
#
# 9.)
# Which services ON THE FIREWALL should be accessible from either the internet
# (or other untrusted networks), the dmz or internal (trusted networks)?
# (see no.13 & 14 if you want to route traffic through the firewall) XXX
#
# Enter all ports or known portnames below, seperated by a space.
# TCP services (e.g. SMTP, WWW) must be set in FW_SERVICES_*_TCP, and
# UDP services (e.g. syslog) must be set in FW_SERVICES_*_UDP.
# e.g. if a webserver on the firewall should be accessible from the internet:
# FW_SERVICES_EXT_TCP="www"
# e.g. if the firewall should receive syslog messages from the dmz:
# FW_SERVICES_DMZ_UDP="syslog"
# For IP protocols (like GRE for PPTP, or OSPF for routing) you need to set
# FW_SERVICES_*_IP with the protocol name or number (see /etc/protocols)
#
# Choice: leave empty or any number of ports, known portnames (from
# /etc/services) and port ranges seperated by a space. Port ranges are
# written like this: allow port 1 to 10 -> "1:10"
# e.g. "", "smtp", "123 514", "3200:3299", "ftp 22 telnet 512:514"
# For FW_SERVICES_*_IP enter the protocol name (like "igmp") or number ("2")
#
# Common: smtp domain
FW_SERVICES_EXT_TCP=""
## Type:        string
# Common: domain
FW_SERVICES_EXT_UDP=""
# Common: domain
## Type:        string
# For VPN/Routing which END at the firewall!!
FW_SERVICES_EXT_IP=""
## Type:        string
# Port numbers of RPC services are dynamically assigned by the portmapper.
# Therefore "rpcinfo -p localhost" is used to automatically determine the
# currently assigned port for the services specified here.
# Typical choice: mountd nfs
FW_SERVICES_EXT_RPC=""
## Type:        string
#
# Common: smtp domain
FW_SERVICES_DMZ_TCP=""
## Type:        string
# Common: domain
FW_SERVICES_DMZ_UDP=""
## Type:        string
# For VPN/Routing which END at the firewall!!
FW_SERVICES_DMZ_IP=""
## Type:        string
# Port numbers of RPC services are dynamically assigned by the portmapper.
# Therefore "rpcinfo -p localhost" is used to automatically determine the
# currently assigned port for the services specified here.
# Typical choice: mountd nfs
FW_SERVICES_DMZ_RPC=""
## Type:        string
#
# Common: ssh smtp domain
```

```
FW_SERVICES_INT_TCP="ssh ntp bootps bootpc xdmcp ipp postgresql cvspserver \
tftp 20 21 sunrpc domain netbios-ns netbios-dgm netbios-ssn microsoft-ds"
## Type:        string
# Common: domain syslog
FW_SERVICES_INT_UDP="ssh ntp bootps bootpc xdmcp ipp postgresql cvspserver \
tftp 20 21 sunrpc domain netbios-ns netbios-dgm netbios-ssn microsoft-ds"
# For VPN/Routing which END at the firewall!!
FW_SERVICES_INT_IP=""
## Type:        string
# Port numbers of RPC services are dynamically assigned by the portmapper.
# Therefore "rpcinfo -p localhost" is used to automatically determine the
# currently assigned port for the services specified here.
# Typical choice: mountd nfs

FW_SERVICES_INT_RPC="mountd nfs"


## Type:        string
# 9a.)
# External services in QUICKMODE.
# This is only used for QUICKMODE (see 1.)!
# (The settings here are similar to section 9.)
# Which services ON THE FIREWALL should be accessible from either the
# internet (or other untrusted networks), i.e. the external interface(s)
# $FW_DEV_EXT
#
# Enter all ports or known portnames below, seperated by a space.
# TCP services (e.g. SMTP, WWW) must be set in FW_SERVICES_QUICK_TCP, and
# UDP services (e.g. syslog) must be set in FW_SERVICES_QUICK_UDP.
# e.g. if a secure shell daemon on the firewall should be accessible from
# the internet:
# FW_SERVICES_QUICK_TCP="ssh"
# e.g. if the firewall should receive isakmp (IPsec) internet:
# FW_SERVICES_QUICK_UDP="isakmp"
# For IP protocols (like IPsec) you need to set
# FW_SERVICES_QUICK_IP="50"
#
# Choice: leave empty or any number of ports, known portnames (from
# /etc/services) and port ranges seperated by a space. Port ranges are
# written like this: allow port 1 to 10 -> "1:10"
# e.g. "", "smtp", "123 514", "3200:3299", "ftp 22 telnet 512:514"
# For FW_SERVICES_*_IP enter the protocol name (like "igmp") or number ("2")
#
# QUICKMODE: TCP services open to external networks (InterNet)
# (Common: ssh smtp)
FW_SERVICES_QUICK_TCP=""
## Type:        string
# QUICKMODE: UDP services open to external networks (InterNet)
# (Common: isakmp)
FW_SERVICES_QUICK_UDP=""
## Type:        string
# QUICKMODE: IP protocols unconditionally open to external networks (InterNet)
# (For VPN firewall that is VPN gateway: 50)

FW_SERVICES_QUICK_IP=""


## Type:        string
#
# 10.)
# Which services should be accessible from trusted hosts/nets?
#
# Define trusted hosts/networks (doesnt matter if they are internal or
# external) and the TCP and/or UDP services they are allowed to use.
# Please note that a trusted host/net is *not* allowed to ping the firewall
# until you set it to allow also icmp!
#
# Choice: leave FW_TRUSTED_NETS empty or any number of computers and/or
# networks, seperated by a space. e.g. "172.20.1.1 172.20.0.0/16"
# Optional, enter a protocol after a comma, e.g. "1.1.1.1,icmp"
# Optional, enter a port after a protocol, e.g. "2.2.2.2,tcp,22"
#

FW_TRUSTED_NETS=""


## Type:        string
#
# 11.)
# How is access allowed to high (unpriviliged [above 1023]) ports?
#
# You may either allow everyone from anyport access to your highports ("yes"),
# disallow anyone ("no"), anyone who comes from a defined port (portnumber or
# known portname) [note that this is easy to circumvent!], or just your
# defined nameservers ("DNS").
# Note that you can't use rpc requests (e.g. rpcinfo, showmount) as root
# from a firewall using this script (well, you can if you include range
# 600:1023 in FW_SERVICES_EXT_UDP ...).
# Please note that with v2.1 "yes" is not mandatory for active FTP from
# the firewall anymore.
#
# Choice: "yes", "no", "DNS", portnumber or known portname,
#         if not set defaults to "no"
#
# Common: "ftp-data", better is "yes" to be sure that everything else works :-(
FW_ALLOW_INCOMING_HIGHPORTS_TCP="no"
## Type:        string
# Common: "DNS" or "domain ntp", better is "yes" to be sure ...
FW_ALLOW_INCOMING_HIGHPORTS_UDP="DNS"


## Type:        yesno
## Default:     yes
#
# 12.)
# Are you running some of the services below?
# They need special attention - otherwise they won
#
# Set services you are running to "yes", all others to "no",
# if not set defaults to "no"
# If you want to offer the below services to your DMZ as well,
# (and not just internally), set the switches below to "dmz",
# if you even want to offer to the world as well, set to "ext"
# instead of "yes" (NOT RECOMMENDED FOR SECURITY REASONS!)
#
FW_SERVICE_AUTODETECT="yes"
# Autodetect the services below when starting
## Type:        yesno
## Default:     no
# If you are running bind/named set to yes. Remember that you have to open
# port 53 (or "domain") as udp/tcp to allow incoming queries.
# Also FW_ALLOW_INCOMING_HIGHPORTS_UDP needs to be "yes"
FW_SERVICE_DNS="yes"
## Type:        yesno
## Default:     no
# if you use dhclient to get an ip address you have to set this to "yes" !
FW_SERVICE_DHCLIENT="no"
## Type:        yesno
## Default:     no
# set to "yes" if this server is a DHCP server
FW_SERVICE_DHCPD="yes"
## Type:        yesno
## Default:     no
# set to "yes" if this server is running squid. You still have to open the
# tcp port 3128 to allow remote access to the squid proxy service.
FW_SERVICE_SQUID="no"
## Type:        yesno
## Default:     no
# set to "yes" if this server is running a samba server. You still have to
# open the tcp port 139 to allow remote access to SAMBA.
FW_SERVICE_SAMBA="yes"


## Type:        string
#
# 13.)
# Which services accessed from the internet should be allowed to the
# dmz (or internal network - if it is not masqueraded)?
# REQUIRES: FW_ROUTE
#
# With this option you may allow access to e.g. your mailserver. The
# machines must have valid, non-private, IP addresses which were assigned to
# you by your ISP. This opens a direct link to your network, so only use
# this option for access to your dmz!!!!
#
# Choice: leave empty (good choice!) or use the following explained syntax
# of forwarding rules, seperated each by a space.
# A forwarding rule consists of 1) source IP/net and 2) destination IP
# seperated by a comma. e.g. "1.1.1.1,2.2.2.2 3.3.3.3/16,4.4.4.4/24"
# Optional is a protocol, seperated by a comma, e.g. "5.5.5.5,6.6.6.6,igmp"
# Optional is a port after the protocol with a comma, e.g. "0/0,0/0,udp,514"
#
FW_FORWARD=""
```

```
# Beware to use this!

## Type:        string
#
# 14.)
# Which services accessed from the internet should be allowed to masqueraded
# servers (on the internal network or dmz)?
# REQUIRES: FW_ROUTE
#
# With this option you may allow access to e.g. your mailserver. The
# machines must be in a masqueraded segment and may not have public IP addesses!
# Hint: if FW_DEV_MASQ is set to the external interface you have to set
# FW_FORWARD from internal to DMZ for the service as well to allow access
# from internal!
#
# Please note that this should *not* be used for security reasons! You are
# opening a hole to your precious internal network. If e.g. the webserver there
# is compromised - your full internal network is compromised!!
#
# Choice: leave empty (good choice!) or use the following explained syntax
# of forward masquerade rules, seperated each by a space.
# A forward masquerade rule consists of 1) source IP/net, 2) the IP to which
# the requests will be forwarded to (in the dmz/intern net), 3) a protocol
# (tcp/udp only!) and 4) destination port, seperated by a comma (","), e.g.
# "4.0.0.0/8,1.1.1.1,tcp,80"
#
# Optional is a port after the destination port, to redirect the request to
# a different destination port on the destination IP, e.g.
# "4.0.0.0/8,1.1.1.1,tcp,80,81"
#
# Optional is an target IP address on which should the masquerading be decided.
# You have to set the optional port option to use this.
#
# Example:
# 200.200.200.0/24,10.0.0.10,tcp,80,81,202.202.202.202
# The class C network 200.200.200.0/24 trying to access 202.202.202.202 port
# 80 will be forwarded to the internal server 10.0.0.10 on port 81.
# Example:
# 200.200.200.0/24,10.0.0.10,tcp,80
# The class C network 200.200.200.0/24 trying to access anything which goes
# through this firewall ill be forwarded to the internal server 10.0.0.10 on
# port 80
#
####FW_FORWARD_MASQ=""
# Beware to use this!
##########################################################
########### INCOMMING CONNECTIONS #######################
##### SEE also FW_MASQ_NETS for outgoing connections #####
##########################################################
## allow incomming:
##         ftp
##         edonkey
## ...traffic
host_dmz_machine="172.16.50.59"
my_dmz_servers_FW_FORWARD_MASQ=\
" \
0/0,$host_dmz_machine,tcp,20 \
0/0,$host_dmz_machine,tcp,21 \
0/0,$host_dmz_machine,tcp,4662 \
0/0,$host_dmz_machine,udp,4666 \
"
##... What ports does the donkey use?
##  It can run over any port. The defaults it uses are:
##  TCP port 4661 to connect to the server (admin).
##  TCP port 4662 to connect to other clients.
##  UDP port 4665 to send messages to servers other than
## the one you are connected to. ...

FW_FORWARD_MASQ="$my_dmz_servers_FW_FORWARD_MASQ"


##########################################################################
## Type:        string
#
# 15.)
# Which accesses to services should be redirected to a localport on the
# firewall machine?
#
# This can be used to force all internal users to surf via your squid proxy,
# or transparently redirect incoming webtraffic to a secure webserver.
#
```

```
# Choice: leave empty or use the following explained syntax of redirecting
# rules, seperated by a space.
# A redirecting rule consists of 1) source IP/net, 2) destination IP/net,
# 3) protocol (tcp or udp) 3) original destination port and 4) local port to
# redirect the traffic to, seperated by a colon. e.g.:
# "10.0.0.0/8,0/0,tcp,80,3128 0/0,172.20.1.1,tcp,80,8080"
# Please note that as 2) destination, you may add '!' in front of the IP/net
# to specify everything EXCEPT this IP/net.
#
FW_REDIRECT=""


## Type:        yesno
## Default:     yes
#
# 16.)
# Which logging level should be enforced?
# You can define to log packets which were accepted or denied.
# You can also the set log level, the critical stuff or everything.
# Note that logging *_ALL is only for debugging purpose ...
#
# Choice: "yes" or "no", if not set FW_LOG_*_CRIT defaults to "yes", and
# FW_LOG_*_ALL defaults to "no"
#
FW_LOG_DROP_CRIT="yes"
## Type:        yesno
## Default:     no
#
FW_LOG_DROP_ALL="no"
## Type:        yesno
## Default:     yes
#
FW_LOG_ACCEPT_CRIT="yes"
## Type:        yesno
## Default:     no
#
FW_LOG_ACCEPT_ALL="no"


## Type:        yesno
## Default:     yes
#
# 17.)
# Do you want to enable additional kernel TCP/IP security features?
# If set to yes, some obscure kernel options are set.
# (icmp_ignore_bogus_error_responses, icmp_echoreply_rate,
#  icmp_destunreach_rate, icmp_paramprob_rate, icmp_timeexeed_rate,
#  ip_local_port_range, log_martians, mc_forwarding, mc_forwarding,
#  rp_filter, routing flush)
# Tip: Set this to "no" until you have verified that you have got a
# configuration which works for you. Then set this to "yes" and keep it
# if everything still works. (It should!) ;-)
#
# Warning: do not set FW_KERNEL_SECURITY and FW_ANTISPOOF to "no" at the same
# time, otherwise you won't have any spoof protection!
#
# Choice: "yes" or "no", if not set defaults to "yes"
#
FW_KERNEL_SECURITY="yes"


## Type:        yesno
## Default:     no
#
# 17a.)
#
# Setup anti-spoofing rules?
# Anti-Spoofing rules shouldn't be necessary with rp_filter set. They only
# cause headaches with dynamic interfaces.
#
# Warning: do not set FW_KERNEL_SECURITY and FW_ANTISPOOF to "no" at the same
# time, otherwise you won't have any spoof protection!
#
FW_ANTISPOOF="no"


# 18.)
# Keep the routing set on, if the firewall rules are unloaded?
# REQUIRES: FW_ROUTE
#
```

```
# If you are using diald, or automatic dialing via ISDN, if packets need
# to be sent to the internet, you need to turn this on. The script will then
# not turn off routing and masquerading when stopped.
# You *might* also need this if you have got a DMZ.
# Please note that this is *insecure*! If you unload the rules, but are still
# connected, you might your internal network open to attacks!
# The better solution is to remove "/sbin/SuSEfirewall2 stop" or
# "/sbin/init.d/firewall stop" from the ip-down script!
#
#
# Choices "yes" or "no", if not set defaults to "no"
#
FW_STOP_KEEP_ROUTING_STATE="no"


## Type:        yesno
## Default:     yes
#
# 19.)
# Allow (or don't) ICMP echo pings on either the firewall or the dmz from
# the internet? The internet option is for allowing the DMZ and the internal
# network to ping the internet.
# REQUIRES: FW_ROUTE for FW_ALLOW_PING_DMZ and FW_ALLOW_PING_EXT
#
# Choice: "yes" or "no", defaults to "no" if not set
#
FW_ALLOW_PING_FW="no"
## Type:        yesno
## Default:     no
#
FW_ALLOW_PING_DMZ="no"
## Type:        yesno
## Default:     no
#
FW_ALLOW_PING_EXT="yes"
##
# END of /etc/sysconfig/SuSEfirewall2
##




#                                                                          #
#--------------------------------------------------------------------------#
#                                                                          #
# EXPERT OPTIONS - all others please don't change these!                   #
#                                                                          #
#--------------------------------------------------------------------------#
#                                                                          #
## Type:        yesno
## Default:     yes
#
# 20.)
# Allow (or don't) ICMP time-to-live-exceeded to be send from your firewall.
# This is used for traceroutes to your firewall (or traceroute like tools).
#
# Please note that the unix traceroute only works if you say "yes" to
# FW_ALLOW_INCOMING_HIGHPORTS_UDP, and windows traceroutes only if you say
# additionally "yes" to FW_ALLOW_PING_FW
#
# Choice: "yes" or "no", if not set defaults to "no"
#
FW_ALLOW_FW_TRACEROUTE="no"


## Type:        yesno
## Default:     yes
#
# 21.)
# Allow ICMP sourcequench from your ISP?
#
# If set to yes, the firewall will notice when connection is choking, however
# this opens yourself to a denial of service attack. Choose your poison.
#
# Choice: "yes" or "no", if not set defaults to "yes"
#
FW_ALLOW_FW_SOURCEQUENCH="yes"
```

```
## Type:        string(yes,no,int,ext,dmz)
## Default:     int
#
# 22.)
# Allow IP Broadcasts?
#
# If set to yes, the firewall will not filter broadcasts by default.
# This is needed e.g. for Netbios/Samba, RIP, OSPF where the broadcast
# option is used.
# If you do not want to allow them however ignore the annoying log entries,
# set FW_IGNORE_FW_BROADCAST to yes.
#
# Choice: "yes" or "no", if not set defaults to "no"
#

FW_ALLOW_FW_BROADCAST="int"


## Type:        string(yes,no,int,ext,dmz)
## Default:     ext
#
# set to yes to suppress log messages for dropped broadcast packets
#

FW_IGNORE_FW_BROADCAST="no"


## Type:        yesno
## Default:     no
#
# 23.)
# Allow same class routing per default?
# REQUIRES: FW_ROUTE
#
# Do you want to allow routing between interfaces of the same class
# (e.g. between all internet interfaces, or all internal network interfaces)
# be default (so without the need setting up FW_FORWARD definitions)?
#
# Choice: "yes" or "no", if not set defaults to "no"
#
FW_ALLOW_CLASS_ROUTING="no"
## Type:        string
#
# 25.)
# Do you want to load customary rules from a file?
#
# This is really an expert option. NO HELP WILL BE GIVEN FOR THIS!
# READ THE EXAMPLE CUSTOMARY FILE AT /etc/sysconfig/scripts/SuSEfirewall2-custom
#
#FW_CUSTOMRULES="/etc/sysconfig/scripts/SuSEfirewall2-custom"

FW_CUSTOMRULES=""


## Type:        yesno
## Default:     no
#
# 26.)
# Do you want to REJECT packets instead of DROPing?
#
# DROPing (which is the default) will make portscans and attacks much
# slower, as no replies to the packets will be sent. REJECTing means, that
# for every illegal packet, a connection reject packet is sent to the
# sender.
#
# Choice: "yes" or "no", if not set defaults to "no"
#

FW_REJECT="no"


## Type:        string
#
# 27.)
# Tuning your upstream a little bit via HTB (Hierarchical Token Bucket)
# for more information about HTB see http://www.lartc.org
#
# If your download collapses while you have a parallel upload,
# this parameter might be an option for you. It manages your
# upload stream and reserves bandwidth for special packets like
# TCP ACK packets or interactive SSH.
# It's a list of devices and maximum bandwidth in kbit.
# For example, the german TDSL account, provides 128kbit/s upstream
# and 768kbit/s downstream. We can only tune the upstream.
```

```
#
# Example:
# If you want to tune a 128kbit/s upstream DSL device like german TDSL set
# the following values:
# FW_HTB_TUNE_DEV="ppp0,125"
# where ppp0 is your pppoe device and 125 stands for 125kbit/s upstream
#
# you might wonder why 125kbit/s and not 128kbit/s. Well practically you'll
# get a better performance if you keep the value a few percent under your
# real maximum upload bandwidth, to prevent the DSL modem from queuing traffic in
# it's own buffers because queing is done by us now.
# So for a 256kbit upstream
#   FW_HTB_TUNE_DEV="ppp0,250"
# might be a better value than "ppp0,256". There is no perfect value for a
# special kind of modem. The perfect value depends on what kind of traffic you
# have on your line but 5% under your maximum upstream might be a good start.
# Everthing else is special fine tuning.
# If you want to know more about the technical background,
# http://tldp.org/HOWTO/ADSL-Bandwidth-Management-HOWTO/
# is a good start
#
FW_HTB_TUNE_DEV="eth0,184"


## Type:        list(no,drop,reject)
## Default:     drop
#
# 28.)
# What to do with IPv6 Packets?
#
# ip6tables is currently not stateful so it's not possible to implement the
# same features as for IPv4. We currently offer three choices:
#
# - no: do not set any IPv6 rules at all. Your Host will allow any IPv6
#   traffic unless you setup your own rules.
#
# - drop: drop all IPv6 packets. This is the default.
#
# - reject: reject all IPv6 packets
#
# Disallowing IPv6 packets may lead to long timeouts when connecting to IPv6
# Adresses. See FW_IPv6_REJECT_OUTGOING to avoid this.
#
FW_IPv6=""


## Type:        yesno
## Default:     yes
#
# 28a.)
# Reject outgoing IPv6 Packets?
#
# Set to yes to avoid timeouts because of dropped IPv6 Packets. This Option
# does only make sense with FW_IPv6 != no
#
FW_IPv6_REJECT_OUTGOING="yes"


## Type:        list(yes,no,int,ext,dmz)
## Default:     no
#
# 29.)
# Trust level of IPsec packets.
#
# The value specifies how much IPsec packets are trusted. 'int', 'ext' or 'dmz'
# are the respective zones. 'yes' is the same as 'int. 'no' means that IPsec
# packets belong to the same zone as the interface they arrive on.
#
# Note: you still need to explicitely allow IPsec traffic.
# Example:
#   FW_IPSEC_TRUST="int"
#   FW_SERVICES_INT_IP="esp"
#   FW_SERVICES_EXT_UDP="isakmp"
#   FW_PROTECT_FROM_INTERNAL="no"
#
FW_IPSEC_TRUST="no"


## Type:        string
#
```

```
# 29a.)
# fwmark used for IPsec packets
#
# Default is 0x1701d. You normally don't need to change that unless you use
# your own mark rules which use the same number already
#
FW_IPSEC_MARK=""

## Type:        string
#
# only change/activate this if you know what you are doing!
FW_LOG=""
```

Here is the resuling rules from the SuSEfirewall2-script that gets feed to iptables:

```
# Generated by iptables-save v1.2.9 on Tue Feb 22 23:46:14 2005
*mangle
:PREROUTING ACCEPT [87773:126881992]
:INPUT ACCEPT [87760:126881303]
:FORWARD ACCEPT [13:689]
:OUTPUT ACCEPT [82401:4438082]
:POSTROUTING ACCEPT [82710:4493395]
-A PREROUTING -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 20 -j TOS --set-tos 0x08
-A PREROUTING -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j TOS --set-tos 0x08
-A PREROUTING -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 80 -j TOS --set-tos 0x08
-A PREROUTING -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 80 -j TOS --set-tos 0x08
-A PREROUTING -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 53 -j TOS --set-tos 0x10
-A PREROUTING -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 53 -j TOS --set-tos 0x10
-A PREROUTING -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 161 -j TOS --set-tos 0x04
-A PREROUTING -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 162 -j TOS --set-tos 0x04
-A PREROUTING -p udp -m udp --dport 514 -j TOS --set-tos 0x04
-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 20 -j TOS --set-tos 0x08
-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j TOS --set-tos 0x08
-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 80 -j TOS --set-tos 0x08
-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 80 -j TOS --set-tos 0x08
-A OUTPUT -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --sport 53 -j TOS --set-tos 0x10
-A OUTPUT -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 53 -j TOS --set-tos 0x10
-A OUTPUT -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 161 -j TOS --set-tos 0x04
-A OUTPUT -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 162 -j TOS --set-tos 0x04
-A OUTPUT -p udp -m udp --dport 514 -j TOS --set-tos 0x04
-A POSTROUTING -o eth0 -p tcp -m length --length 0:64 -j MARK --set-mark 0xa
-A POSTROUTING -o eth0 -p tcp -m tos --tos Minimize-Delay -m tcp --dport 22 -j MARK --set-mark 0xa
-A POSTROUTING -o eth0 -p tcp -m tos --tos Minimize-Delay -m tcp --sport 22 -j MARK --set-mark 0xa
-A POSTROUTING -o eth0 -p udp -m udp --dport 53 -j MARK --set-mark 0xa
-A POSTROUTING -o eth0 -p tcp -m tcp --dport 53 -j MARK --set-mark 0xa
-A POSTROUTING -o eth0 -p esp -j MARK --set-mark 0xb
COMMIT
# Completed on Tue Feb 22 23:46:14 2005
# Generated by iptables-save v1.2.9 on Tue Feb 22 23:46:14 2005
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:forward_dmz - [0:0]
:forward_ext - [0:0]
:forward_int - [0:0]
:input_dmz - [0:0]
:input_ext - [0:0]
:input_int - [0:0]
:reject_func - [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -d 255.255.255.255 -i eth2 -p udp -m state --state NEW,ESTABLISHED -m udp --sport 68 --dport 67 -j ACCEPT
-A INPUT -i eth2 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 137:138 -j ACCEPT
-A INPUT -d 255.255.255.255 -i eth0 -j DROP
-A INPUT -d 10.0.0.255 -i eth0 -j DROP
-A INPUT -d 255.255.255.255 -i eth1 -j DROP
-A INPUT -d 172.16.50.255 -i eth1 -j DROP
-A INPUT -i eth0 -j input_ext
-A INPUT -i eth1 -j input_dmz
-A INPUT -i eth2 -j input_int
-A INPUT -d 10.0.0.4 -i eth2 -j LOG --log-prefix "SFW2-IN-ACC_DENIED_INT " --log-tcp-options --log-ip-options
-A INPUT -d 10.0.0.4 -i eth2 -j DROP
-A INPUT -j LOG --log-prefix "SFW2-IN-ILL-TARGET " --log-tcp-options --log-ip-options
-A INPUT -j DROP
-A FORWARD -p tcp -m tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu
-A FORWARD -i eth2 -o eth2 -j ACCEPT
-A FORWARD -i eth0 -o eth0 -j ACCEPT
-A FORWARD -i eth1 -o eth1 -j ACCEPT
-A FORWARD -i eth0 -j forward_ext
-A FORWARD -i eth1 -j forward_dmz
-A FORWARD -i eth2 -j forward_int
-A FORWARD -j LOG --log-prefix "SFW2-FWD-ILL-ROUTING " --log-tcp-options --log-ip-options
-A FORWARD -j DROP
-A FORWARD -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -j LOG --log-prefix "SFW2-FORWARD-ERROR " --log-tcp-options --log-ip-options
-A OUTPUT -o lo -j ACCEPT
-A OUTPUT -p icmp -m icmp --icmp-type 11 -j LOG --log-prefix "SFW2-OUT-TRACERT-ATTEMPT " --log-tcp-options --log-ip-options
-A OUTPUT -p icmp -m icmp --icmp-type 11 -j DROP
-A OUTPUT -p icmp -m icmp --icmp-type 3/4 -j ACCEPT
-A OUTPUT -p icmp -m icmp --icmp-type 3/9 -j ACCEPT
-A OUTPUT -p icmp -m icmp --icmp-type 3/10 -j ACCEPT
-A OUTPUT -p icmp -m icmp --icmp-type 3/13 -j ACCEPT
-A OUTPUT -p icmp -m icmp --icmp-type 3 -j DROP
-A OUTPUT -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -j LOG --log-prefix "SFW2-OUTPUT-ERROR " --log-tcp-options --log-ip-options
-A forward_dmz -d 192.168.100.100 -j LOG --log-prefix "SFW2-FWDdmz-DROP-CIRCUMV " --log-tcp-options --log-ip-options
-A forward_dmz -d 192.168.100.100 -j DROP
-A forward_dmz -d 10.0.0.4 -j LOG --log-prefix "SFW2-FWDdmz-DROP-CIRCUMV " --log-tcp-options --log-ip-options
-A forward_dmz -d 10.0.0.4 -j DROP
-A forward_dmz -o eth0 -p icmp -m state --state NEW -m icmp --icmp-type 8 -j ACCEPT
```

```
-A forward_dmz -p icmp -m state --state RELATED -m icmp --icmp-type 3 -j ACCEPT
-A forward_dmz -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 0 -j ACCEPT
-A forward_dmz -s 192.168.100.0/255.255.255.0 -o eth0 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A forward_dmz -d 192.168.100.0/255.255.255.0 -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A forward_dmz -s 172.16.50.59 -d 212.242.40.0/255.255.255.0 -o eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 53 -
j ACCEPT
-A forward_dmz -s 212.242.40.0/255.255.255.0 -d 172.16.50.59 -i eth0 -p udp -m state --state RELATED,ESTABLISHED -m udp --sport 53 -j ACCEPT
-A forward_dmz -s 172.16.50.59 -d 84.243.240.2 -o eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 123 -j ACCEPT
-A forward_dmz -s 84.243.240.2 -d 172.16.50.59 -i eth0 -p udp -m state --state RELATED,ESTABLISHED -m udp --sport 123 -j ACCEPT
-A forward_dmz -s 172.16.50.59 -d 192.38.95.24 -o eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 22 -j ACCEPT
-A forward_dmz -s 192.38.95.24 -d 172.16.50.59 -i eth0 -p tcp -m state --state RELATED,ESTABLISHED -m tcp --sport 22 -j ACCEPT
-A forward_dmz -s 172.16.50.59 -o eth0 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A forward_dmz -d 172.16.50.59 -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A forward_dmz -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 20 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDdmz-ACC-REVMASQ " --
log-tcp-options --log-ip-options
-A forward_dmz -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j ACCEPT
-A forward_dmz -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 21 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDdmz-ACC-REVMASQ " --
log-tcp-options --log-ip-options
-A forward_dmz -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 21 -j ACCEPT
-A forward_dmz -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 4662 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDdmz-ACC-
REVMASQ " --log-tcp-options --log-ip-options
-A forward_dmz -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 4662 -j ACCEPT
-A forward_dmz -d 172.16.50.59 -i eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 4666 -j ACCEPT
-A forward_dmz -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_dmz -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-FWDdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_dmz -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-FWDdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_dmz -p icmp -m icmp --icmp-type 8 -j LOG --log-prefix "SFW2-FWDdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_dmz -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-FWDdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_dmz -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-FWDdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_dmz -p udp -j LOG --log-prefix "SFW2-FWDdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_dmz -m state --state INVALID -j LOG --log-prefix "SFW2-FWDdmz-DROP-DEFLT-INV " --log-tcp-options --log-ip-options
-A forward_dmz -j DROP
-A forward_ext -d 192.168.100.100 -j LOG --log-prefix "SFW2-FWDext-DROP-CIRCUMV " --log-tcp-options --log-ip-options
-A forward_ext -d 192.168.100.100 -j DROP
-A forward_ext -d 172.16.50.50 -j LOG --log-prefix "SFW2-FWDext-DROP-CIRCUMV " --log-tcp-options --log-ip-options
-A forward_ext -d 172.16.50.50 -j DROP
-A forward_ext -p icmp -m state --state ESTABLISHED -m icmp --icmp-type 0 -j ACCEPT
-A forward_ext -p icmp -m state --state RELATED -m icmp --icmp-type 3 -j ACCEPT
-A forward_ext -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 0 -j ACCEPT
-A forward_ext -s 192.168.100.0/255.255.255.0 -o eth0 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A forward_ext -d 192.168.100.0/255.255.255.0 -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A forward_ext -s 172.16.50.59 -d 212.242.40.0/255.255.255.0 -o eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 53 -
j ACCEPT
-A forward_ext -s 212.242.40.0/255.255.255.0 -d 172.16.50.59 -i eth0 -p udp -m state --state RELATED,ESTABLISHED -m udp --sport 53 -j ACCEPT
-A forward_ext -s 172.16.50.59 -d 84.243.240.2 -o eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 123 -j ACCEPT
-A forward_ext -s 84.243.240.2 -d 172.16.50.59 -i eth0 -p udp -m state --state RELATED,ESTABLISHED -m udp --sport 123 -j ACCEPT
-A forward_ext -s 172.16.50.59 -d 192.38.95.24 -o eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 22 -j ACCEPT
-A forward_ext -s 192.38.95.24 -d 172.16.50.59 -i eth0 -p tcp -m state --state RELATED,ESTABLISHED -m tcp --sport 22 -j ACCEPT
-A forward_ext -s 172.16.50.59 -o eth0 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A forward_ext -d 172.16.50.59 -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A forward_ext -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 20 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDext-ACC-REVMASQ " --
log-tcp-options --log-ip-options
-A forward_ext -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j ACCEPT
-A forward_ext -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 21 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDext-ACC-REVMASQ " --
log-tcp-options --log-ip-options
-A forward_ext -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 21 -j ACCEPT
-A forward_ext -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 4662 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDext-ACC-
REVMASQ " --log-tcp-options --log-ip-options
-A forward_ext -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 4662 -j ACCEPT
-A forward_ext -d 172.16.50.59 -i eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 4666 -j ACCEPT
-A forward_ext -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_ext -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-FWDext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_ext -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-FWDext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_ext -p icmp -m icmp --icmp-type 8 -j LOG --log-prefix "SFW2-FWDext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_ext -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-FWDext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_ext -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-FWDext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_ext -p udp -j LOG --log-prefix "SFW2-FWDext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_ext -m state --state INVALID -j LOG --log-prefix "SFW2-FWDext-DROP-DEFLT-INV " --log-tcp-options --log-ip-options
-A forward_ext -j DROP
-A forward_int -d 10.0.0.4 -j LOG --log-prefix "SFW2-FWDint-DROP-CIRCUMV " --log-tcp-options --log-ip-options
-A forward_int -d 10.0.0.4 -j DROP
-A forward_int -d 172.16.50.50 -j LOG --log-prefix "SFW2-FWDint-DROP-CIRCUMV " --log-tcp-options --log-ip-options
-A forward_int -d 172.16.50.50 -j DROP
-A forward_int -o eth0 -p icmp -m state --state NEW -m icmp --icmp-type 8 -j ACCEPT
-A forward_int -p icmp -m state --state RELATED -m icmp --icmp-type 3 -j ACCEPT
-A forward_int -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 0 -j ACCEPT
-A forward_int -s 192.168.100.0/255.255.255.0 -o eth0 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A forward_int -d 192.168.100.0/255.255.255.0 -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A forward_int -s 172.16.50.59 -d 212.242.40.0/255.255.255.0 -o eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 53 -
j ACCEPT
-A forward_int -s 212.242.40.0/255.255.255.0 -d 172.16.50.59 -i eth0 -p udp -m state --state RELATED,ESTABLISHED -m udp --sport 53 -j ACCEPT
-A forward_int -s 172.16.50.59 -d 84.243.240.2 -o eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 123 -j ACCEPT
-A forward_int -s 84.243.240.2 -d 172.16.50.59 -i eth0 -p udp -m state --state RELATED,ESTABLISHED -m udp --sport 123 -j ACCEPT
-A forward_int -s 172.16.50.59 -d 192.38.95.24 -o eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 22 -j ACCEPT
-A forward_int -s 192.38.95.24 -d 172.16.50.59 -i eth0 -p tcp -m state --state RELATED,ESTABLISHED -m tcp --sport 22 -j ACCEPT
-A forward_int -s 172.16.50.59 -o eth0 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A forward_int -d 172.16.50.59 -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A forward_int -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 20 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDint-ACC-REVMASQ " --
log-tcp-options --log-ip-options
-A forward_int -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j ACCEPT
-A forward_int -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 21 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDint-ACC-REVMASQ " --
log-tcp-options --log-ip-options
-A forward_int -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 21 -j ACCEPT
-A forward_int -d 172.16.50.59 -i eth0 -p tcp -m tcp --dport 4662 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDint-ACC-
REVMASQ " --log-tcp-options --log-ip-options
-A forward_int -d 172.16.50.59 -i eth0 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 4662 -j ACCEPT
-A forward_int -d 172.16.50.59 -i eth0 -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 4666 -j ACCEPT
-A forward_int -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-FWDint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_int -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-FWDint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_int -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-FWDint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_int -p icmp -m icmp --icmp-type 8 -j LOG --log-prefix "SFW2-FWDint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_int -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-FWDint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_int -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-FWDint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_int -p udp -j LOG --log-prefix "SFW2-FWDint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A forward_int -m state --state INVALID -j LOG --log-prefix "SFW2-FWDint-DROP-DEFLT-INV " --log-tcp-options --log-ip-options
-A forward_int -j DROP
-A input_dmz -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 0 -j ACCEPT
```

```
-A input_dmz -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 3 -j ACCEPT
-A input_dmz -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 11 -j ACCEPT
-A input_dmz -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 12 -j ACCEPT
-A input_dmz -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 14 -j ACCEPT
-A input_dmz -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 18 -j ACCEPT
-A input_dmz -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-INdmz-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-INdmz-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-INdmz-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-INdmz-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 2 -j LOG --log-prefix "SFW2-INdmz-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -j DROP
-A input_dmz -p tcp -m tcp --dport 113 --tcp-flags SYN,RST,ACK SYN -j reject_func
-A input_dmz -p tcp -m tcp --dport 21 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 21 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 22 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 22 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 53 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 53 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 111 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 111 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 139 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 139 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 445 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 445 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 726 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 726 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 901 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 901 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 1033 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 1033 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m tcp --dport 2049 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP " --log-tcp-options --log-ip-options
-A input_dmz -p tcp -m tcp --dport 2049 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_dmz -p tcp -m state --state RELATED,ESTABLISHED -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-ACC-HiTCP " --
log-tcp-options --log-ip-options
-A input_dmz -p tcp -m state --state RELATED,ESTABLISHED -j ACCEPT
-A input_dmz -p udp -m state --state RELATED,ESTABLISHED -j ACCEPT
-A input_dmz -p udp -m udp --dport 21 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 22 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 53 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 53 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 69 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 111 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 111 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 137 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 137 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 138 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 138 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 139 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 445 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 723 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 726 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 901 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 1027 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 1033 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 2049 -m state --state NEW -j DROP
-A input_dmz -p udp -m udp --dport 2049 -m state --state NEW -j DROP
-A input_dmz -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-INdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-INdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 8 -j LOG --log-prefix "SFW2-INdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-INdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_dmz -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-INdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_dmz -p udp -j LOG --log-prefix "SFW2-INdmz-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_dmz -m state --state INVALID -j LOG --log-prefix "SFW2-INdmz-DROP-DEFLT-INV " --log-tcp-options --log-ip-options
-A input_dmz -j DROP
-A input_ext -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-INext-ACC-SOURCEQUENCH " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 4 -j ACCEPT
-A input_ext -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 0 -j ACCEPT
-A input_ext -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 3 -j ACCEPT
-A input_ext -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 11 -j ACCEPT
-A input_ext -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 12 -j ACCEPT
-A input_ext -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 14 -j ACCEPT
-A input_ext -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 18 -j ACCEPT
-A input_ext -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-INext-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-INext-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-INext-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-INext-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 2 -j LOG --log-prefix "SFW2-INext-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -j DROP
-A input_ext -p tcp -m tcp --dport 113 --tcp-flags SYN,RST,ACK SYN -j reject_func
-A input_ext -p tcp -m tcp --dport 21 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 21 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 22 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 22 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 53 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 53 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 111 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 111 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 139 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 139 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 445 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 445 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 726 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 726 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 901 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 901 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 1033 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 1033 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_ext -p tcp -m tcp --dport 2049 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP " --log-tcp-options --log-ip-options
-A input_ext -p tcp -m tcp --dport 2049 --tcp-flags SYN,RST,ACK SYN -j DROP
```

```
-A input_ext -p tcp -m state --state RELATED,ESTABLISHED -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-ACC-HiTCP " --
log-tcp-options --log-ip-options
-A input_ext -p tcp -m state --state RELATED,ESTABLISHED -j ACCEPT
-A input_ext -p udp -m state --state RELATED,ESTABLISHED -j ACCEPT
-A input_ext -p udp -m udp --dport 21 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 22 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 53 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 53 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 69 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 111 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 111 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 137 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 137 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 138 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 138 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 139 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 445 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 723 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 726 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 901 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 1027 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 1033 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 2049 -m state --state NEW -j DROP
-A input_ext -p udp -m udp --dport 2049 -m state --state NEW -j DROP
-A input_ext -p udp -m state --state ESTABLISHED -m udp --dport 61000:65095 -j ACCEPT
-A input_ext -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 8 -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_ext -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_ext -p udp -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_ext -m state --state INVALID -j LOG --log-prefix "SFW2-INext-DROP-DEFLT-INV " --log-tcp-options --log-ip-options
-A input_ext -j DROP
-A input_int -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 0 -j ACCEPT
-A input_int -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 3 -j ACCEPT
-A input_int -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 11 -j ACCEPT
-A input_int -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 12 -j ACCEPT
-A input_int -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 14 -j ACCEPT
-A input_int -p icmp -m state --state RELATED,ESTABLISHED -m icmp --icmp-type 18 -j ACCEPT
-A input_int -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-INint-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-INint-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-INint-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-INint-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 2 -j LOG --log-prefix "SFW2-INint-DROP-ICMP-CRIT " --log-tcp-options --log-ip-options
-A input_int -p icmp -j DROP
-A input_int -p tcp -m tcp --dport 22 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 22 -j ACCEPT
-A input_int -p tcp -m tcp --dport 123 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 123 -j ACCEPT
-A input_int -p tcp -m tcp --dport 67 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 67 -j ACCEPT
-A input_int -p tcp -m tcp --dport 68 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 68 -j ACCEPT
-A input_int -p tcp -m tcp --dport 69 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 69 -j ACCEPT
-A input_int -p tcp -m tcp --dport 20 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 20 -j ACCEPT
-A input_int -p tcp -m tcp --dport 21 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 21 -j ACCEPT
-A input_int -p tcp -m tcp --dport 111 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 111 -j ACCEPT
-A input_int -p tcp -m tcp --dport 53 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 53 -j ACCEPT
-A input_int -p tcp -m tcp --dport 137 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 137 -j ACCEPT
-A input_int -p tcp -m tcp --dport 138 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 138 -j ACCEPT
-A input_int -p tcp -m tcp --dport 139 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 139 -j ACCEPT
-A input_int -p tcp -m tcp --dport 445 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-TCP " --log-tcp-options --log-ip-
options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 445 -j ACCEPT
-A input_int -p udp -m state --state NEW -m udp --dport 2049 -j LOG --log-prefix "SFW2-INint-ACC-RPC " --log-tcp-options --log-ip-options
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 2049 -j ACCEPT
-A input_int -p udp -m state --state NEW -m udp --dport 723 -j LOG --log-prefix "SFW2-INint-ACC-RPC " --log-tcp-options --log-ip-options
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 723 -j ACCEPT
-A input_int -p tcp -m state --state NEW -m tcp --dport 726 -j LOG --log-prefix "SFW2-INint-ACC-RPC " --log-tcp-options --log-ip-options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 726 -j ACCEPT
-A input_int -p tcp -m state --state NEW -m tcp --dport 2049 -j LOG --log-prefix "SFW2-INint-ACC-RPC " --log-tcp-options --log-ip-options
-A input_int -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 2049 -j ACCEPT
-A input_int -p tcp -m tcp --dport 113 --tcp-flags SYN,RST,ACK SYN -j reject_func
-A input_int -p tcp -m tcp --dport 22 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 22 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_int -p tcp -m tcp --dport 53 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 53 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_int -p tcp -m tcp --dport 111 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 111 --tcp-flags SYN,RST,ACK SYN -j DROP
```

```
-A input_int -p tcp -m tcp --dport 139 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 139 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_int -p tcp -m tcp --dport 445 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 445 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_int -p tcp -m tcp --dport 726 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 726 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_int -p tcp -m tcp --dport 901 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 901 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_int -p tcp -m tcp --dport 1033 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 1033 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_int -p tcp -m tcp --dport 2049 --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP " --log-tcp-options --log-ip-options
-A input_int -p tcp -m tcp --dport 2049 --tcp-flags SYN,RST,ACK SYN -j DROP
-A input_int -p tcp -m state --state RELATED,ESTABLISHED -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-ACC-HiTCP " --
log-tcp-options --log-ip-options
-A input_int -p tcp -m state --state RELATED,ESTABLISHED -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 22 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 123 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 67 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 68 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 69 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 20 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 21 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 111 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 53 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 137 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 138 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 139 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 445 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 1024 -j ACCEPT
-A input_int -p udp -m state --state NEW,RELATED,ESTABLISHED -m udp --dport 1025 -j ACCEPT
-A input_int -p udp -m state --state RELATED,ESTABLISHED -j ACCEPT
-A input_int -p udp -m udp --dport 22 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 53 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 53 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 53 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 67 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 69 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 111 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 111 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 123 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 137 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 137 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 138 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 138 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 139 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 445 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 723 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 726 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 901 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 1027 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 1033 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 2049 -m state --state NEW -j DROP
-A input_int -p udp -m udp --dport 2049 -m state --state NEW -j DROP
-A input_int -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 4 -j LOG --log-prefix "SFW2-INint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 5 -j LOG --log-prefix "SFW2-INint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 8 -j LOG --log-prefix "SFW2-INint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 13 -j LOG --log-prefix "SFW2-INint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_int -p icmp -m icmp --icmp-type 17 -j LOG --log-prefix "SFW2-INint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_int -p udp -j LOG --log-prefix "SFW2-INint-DROP-DEFLT " --log-tcp-options --log-ip-options
-A input_int -m state --state INVALID -j LOG --log-prefix "SFW2-INint-DROP-DEFLT-INV " --log-tcp-options --log-ip-options
-A input_int -j DROP
-A reject_func -p tcp -j REJECT --reject-with tcp-reset
-A reject_func -p udp -j REJECT --reject-with icmp-port-unreachable
-A reject_func -j REJECT --reject-with icmp-proto-unreachable
COMMIT
# Completed on Tue Feb 22 23:46:14 2005
# Generated by iptables-save v1.2.9 on Tue Feb 22 23:46:14 2005
*nat
:PREROUTING ACCEPT [198:14820]
:POSTROUTING ACCEPT [443:44400]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -d 10.0.0.4 -i eth0 -p tcp -m tcp --dport 20 -j DNAT --to-destination 172.16.50.59:20
-A PREROUTING -d 10.0.0.4 -i eth0 -p tcp -m tcp --dport 21 -j DNAT --to-destination 172.16.50.59:21
-A PREROUTING -d 10.0.0.4 -i eth0 -p tcp -m tcp --dport 4662 -j DNAT --to-destination 172.16.50.59:4662
-A PREROUTING -d 10.0.0.4 -i eth0 -p udp -m udp --dport 4666 -j DNAT --to-destination 172.16.50.59:4666
-A POSTROUTING -s 192.168.100.0/255.255.255.0 -o eth0 -j MASQUERADE
-A POSTROUTING -s 172.16.50.59 -d 212.242.40.0/255.255.255.0 -o eth0 -p udp -m udp --dport 53 -j MASQUERADE --to-ports 61000-65095
-A POSTROUTING -s 172.16.50.59 -d 84.243.240.2 -o eth0 -p udp -m udp --dport 123 -j MASQUERADE --to-ports 61000-65095
-A POSTROUTING -s 172.16.50.59 -d 192.38.95.24 -o eth0 -p tcp -m tcp --dport 22 -j MASQUERADE --to-ports 61000-65095
-A POSTROUTING -s 172.16.50.59 -o eth0 -j MASQUERADE
COMMIT
# Completed on Tue Feb 22 23:46:14 2005
```

The lengthy script (1000 lines) and output (500 lines) is just one configuration.

Appendix B

Timetable

The project did follow a plan, and as it progressed risks where assesed along the way. As it was discovered what was to be created, some rough estimates where made. Each estimate is roughly the length of *a handful*[1].

Its a psychological thing and means that, the human comprehension is believed to be good at grasping up to five 'things' at a time - i.e. 5 hours of work (a day), 5 days, 4 weeks (one month) - or five classes on a diagram, and so on.

B.1 Project schedule

As always, any software-plan should be multiplied with *Phi* (3,14).

Why? Because programmers only estimate how long it takes to code something (1 time-unit). But the thing needs to be designed too (1 time-unit) and tested (1 time-unit), and some administrative overhead is always present (0.14 time-units) - and that all accumulates to: 1+1+1+0.14=3.14.

In our case the time-encoded estimate becomes: 2 months of specification and design, 2 months of implementation and 2 months of testing and report-writing - or roughly: 2+2+2+overhead = 7 months.

The topics to be covered, are:

- One month of Linux firewalls: area research, i.e. commands, configurations, howto, try-outs
- One month of Databases: common database theory, SQL, ER-diagrams, design, MySQL/Postgress-opportunities, -usage, -try-outs
- One month of KDE & Qt: API, classes, opportunities, build- and develop-environment, component-model, design, try-outs
- Two months of implementation of end-user-GUI-application: Framework/skeleton, parts for config, database-interaction (SQL), iptables-interaction (parser/generator), rule-editor and -manipulation (rule-table-view),
- One month of Linux kernel- and netfilter-modules: howto, design, try-outs and implementation
- One month of report-writing.

The above rough estimates gives 7 months worth of project work. Some of it have overlapping areas, like the implementation of the GUI: undoubtly, some new knowledge about KDE/Qt and/or the database-design will surface.

[1]That was once taught to me - at a time-estimation-course at DELTA-institute in Hørsholm.

B.2 Project progress diary

At this stage, a plan exists (really!). With, a) the initial motivation in place, b) background and domains drawn up, c) statement of requirements, d) the grand creation of a system design along with e) a time-schedule to implement it - it is now time to report, how progress and adherence to the mighty plan went.

Adjustments will follow as the project progresses as indicated in Sec. 3. Following is a chronological list of intermediate milestones reached, indicating how the project progresses and when the plan was adjusted:

01-02-2005 Startup of project. Makes initial plans for rough layout of tasks ahead. Startup on report disposition/contents.

07-02-2005 Initial description of foundation and background of project and it's domain.

21-02-2005 Requirements descriptions initiated

30-02-2005 System design initiated.

18-03-2005 Start on the first development prototype - a frame work which can demonstrate the System Design of parsing from the sources of proc, ifconfig and netfilter/iptables into the database, and show some simple extracts in a Gui with the dynamic features.

25-03-2005 Introduction, background and requirements done.

30-03-2005 A week of infection - a real virus this time (influenza).

07-04-2005 Database design and SQL study. Trying out MySQL.

21-04-2005 Study of Qt-, KDE-, SQL-libs and gnu-build-tools. Got initial database relations designed and table layed out. Switched to Postgress and dropped MySQL.

27-04-2005 Simple proof-of-principles-tests done for SQL-usage, KDE-integrations,

04-05-2005 Got independently build module-structure up and running using KParts ('dynamic dialogs' within KDE). These do not need re-building/-linking to the main app.

15-05-2005 Got KCM-parts for configuration up and running.

20-05-2005 Lots of debugging: the string- and template-based naming of the Factory-system (DSO's, loading of libraries and execution of factories - generic and custom), the autoconf/automake-system of naming and placement, the placement and naming of and in the KDE-hierarchy of dirs, libs and desktop-files.

27-05-2005 Got SQL-db and ipt_cmd components up and running.

08-06-2005 Parser for iptable-save-output done.

13-06-2005 Parser re-factored (re-written/-sectionised). Generator done.

22-06-2005 Rule-list-view well underway.

06-07-2005 Root-sub-shell-communications done.
Rule-listview has many issues:
Speed and interactivity of SQL-query-overhead versus Item-iteration (ruleview has become cache of database, we need SQL-capabilities in the listview: searching, sorting, iterating, change-control etc.),
Enforced sorting of listview-items,
Finding and searching items,
DnD-handling and -indicators,

GUI-presentation, -drawing, rendering- and update-speed (signals and items: hiding listview-items and logging),
Reading out-of-order entries (Re-sorting of orphans),
Committing to database (INSERT/UPDATE/DELETE-constructs of SQL-statement-stack), and just plain regular bugs.

13-07-2005 Rule-listview finally in roundtrip-working state. Cannot merge concurrent DB-changes with Listview-changes yet - so meanwhile, locking of tables is done as remedy.

20-07-2005 Proc-listview up and running.

25-07-2005 Kernel-QUEUE-target-handler finished. Can grab packets and later on accept/drop any particular one on any order. Integrated into the ProcView.

27-07-2005 DCOP-rule-manipulation done. Allows ProcView (and any other DCOP-client e.g. shell-script) to manipulate the rule-sets.

31-07-2005 Quick-setup-wizard initiated, for making the rules allowing our ProcView to make rules for connections and processes.

03-08-2005 End of development - closing the code, and start of report writing. The Quick-setup-wizard don't work right, and the rule-search and -injection from the ProcView and Quick-setup-wizard don't work either: the DCOP-interface of the RuleView needs a change to the insertNode()-function in the DCOP-interface.
Principle of rule-manipulation though DCOP works fine though, and packet capture with manual verdicts are working fine too.

Appendix C

User guide

I have decided *not* to make a user-guide - and settle with a live demonstration of the program. There will be a severe impact in how the user initially will use the program, due to the setup-problems still pending.

Expertise is still needed in setting up Postgres (and creating working install-rpm's), and in creating an initial set of rules (SetupWizard). This is likely to change soon, but makes it less relevant to make a user-guide showing how to get started. The daily usage of the program is mostly reflected in the descriptions and screen-shots of Sec. 7.5, and would simply be repeated here in more elaborate detail.

A user-guide *could* be completed and presented *before* the actual program was made. But our chosen strategy of using an agile-approach suggests, that it should be postponed until actually needed - and it isn't in this report. Furthermore, a user-guide is most feasible in the Online-help of a program - and while initially present, the Online-help wasn't completed either.

Overall, a live demonstration will show all the issues much better - what works, as well as what doesn't.

Bibliography

[lpfw] The source-code and binaries of this project:
http://www.sourceforge.net/projects/lpfw and http://lpfw.sourceforge.net
http://www.freshmeat.net
Search for 'LPFW' and the source and announcement should be at your disposal - as of late Aug./early Sep. 2005.

[netfilter] netfilter/iptables: packetfilter-firewall-framework for Linux-kernel V2.4 onwards
http://www.netfilter.org/

[KDERef] KDE/Qt: API, Installation and usage of KDE, Qt and KDevelop (automake/autoconf):
http://docs.kde.org/en/3.4/...
http://developer.kde.org/documentation/...
Look at the Howto- and Tutorial-sections to find out which KDE-Classes are for your purpuse - and look in the source-code of each KDE-Class to learn how to *really* use it! (I suspect that's why its included in the HTML-Ref-Docs...)
http://doc.trolltech.com/...
http://www.digitalfanatics.org/projects/qt_tutorial/index.html
Again the Tutorial- and Module-sections are the place to get an overview of usage.
http://webcvs.kde.org//%7Echeckout%7E/kdelibs/kio/DESKTOP_ENTRY_STANDARD
http://www.freedesktop.org/wiki/Standards
The installation-layout and contents of '.desktop'-files
http://kdevelop.org/...
http://women.kde.org/articles/tutorials/kdevelop3/index.html
http://www.kdevelop.org/index.html?filename=3.0/doc/tutorial_autoconf.html
You'll need to build eventually, and using these links - are how *not* to loose your sanity while trying...
And loads more - too many to list - and they are a moving target (some dead, some moved). The source and how to use it, lives on the net - it's difficult to list comprehensivly.
Many links are not up to date and contain irrelevant, obsolete and/or errors in the infomation, but some of the infomation is good! - just cross-correlate the various tutorials, and you'll get the essence just about right. The key issue is: READ MORE THAN ONE!!! - they aren't sufficient individually...

[SW-Eng.(7ed)] Software Engineering (7 Ed. 2004), Ian Sommerville, Addison Wesley, ISBN 0-321-21026-3
Good allround picking in Chap 2 - 5, 8, 11 - and some others (mostly general terms, not specific solutions)

[ERD-Intro] **E**ntity-**R**elationship (ER) diagrams for **D**ummies:
"... an informal introduction to data modeling using the Entity-Relationship (ER) approach. It is intended for someone who is familiar with relational databases but

who has no experience in data modeling..."
http://www.utexas.edu/its/windows/database/datamodeling/index.html

[DBSystems] Database Systems: The complete book (2001), Garcia-Molina, Jeffrey D. Ullman and Jennifer Widom, Prentice Hall, ISBN 0-13-031995-3
Most of Chapters 2 through 8 - except Chap 5 (too much theory).

[ApplComms] Applied Data Communications and Networks (1996), W. Buchanan, Chapman & Hall, ISBN 0-412-75430-4
Loosely Chap 1, 2, 3 and 5 - forget the rest (unless you want to be an electrical-engineer).

[MS-Survey] Microsoft Survey
(in English) http://www.webuser.co.uk/news/60751.html
(in Danish) http://www.virus112.dk/dk/news/microsoft/microsoft_bliv_mere_sikker.html

[LinuxFWip] Linux Firewalls, Second Edition (2001), Robert L. Ziegeler, New Riders, ISBN 0-7357-1099-6
Only Chap 1 - 8 are about firewall-types, and netfilter/iptables in particular (the rest is about general network security - use [WilyHacker] instead).

[SecCompPP] Security in Computing (2002), C. P. Pfleeger, S. L. Pfleeger, Addison Wesley, ISBN 0-13-035548-8
Mostly backgound for general (network) security, firewall-types and do'es-and-don'ts (used for context, arguments and a few quotes).

[WilyHacker] Firewalls and Internet Security - Repelling the Wily Hacker (2001), Second Edition, W. R. Cheswich, S. M. Bellovin, A. D. Rubin, Addison Wesley, ISBN 0-201-63466-X
Allround here and there - no Chapter in particular.

[CPPRefBS] The C++ Programming Language (1997), Third Ed., Bjarne Stroustrup, Addison Wesley, ISBN 0-201-88954-4
THE Bible of C++ (from the man him self).

[IntroLinProg] Beginning Linux Programming, Third Ed. (2004), Neil Matthew, Richard Stones, Wiley (Wrox), ISBN 0-7645-4497-7
Allround hands-on intro to Linux/Unix-system-level C-programming: Processes (child-parent, root-rights), Sockets, Pipes, File- and Memory-handles, Terminal-IO, Shells, Devices, Kernel-modules and programming utils (debuggers, mem-leak-detectors, makefiles).

[Firmato] Firmato: A novel firewall management toolkit.
Source: ACM Transactions on Computer Systems (TOCS) archive Volume 22 , Issue 4 (November 2004) - ISSN:0734-2071
Authors: Yair Bartal, Alain Mayer, Kobbi Nissim, Avishai Wool.

[Fang] Fang: A Firewall Analysis Engine.
Source: SP archive Proceedings of the 2000 IEEE Symposium on Security and Privacy (Year 2000) - ISBN:0-7695-0665-8
Authors: Alain Mayer, Avishai Wool, Elisha Ziskind.

[ITVal] ITVal: A Firewall Analysis Engine. (V0.3)
Source: USENIX 2005 Annual Technical Conference, FREENIX Track (Year 2005) - Note: The paper has restricted access until spring-2006 - use google's cache-link instead, to circumvent the password-query on the link of the article.
Authors: Robert Marmorstein, Phil Kearns.
http://www.cs.wm.edu/~rmmarm/ITVal/index.html