RHODES UNIVERSITY

Where leaders learn

Automated Firewall Rule Set Generation Through Passive Traffic Inspection

Georg-Christian Pranschke

November 11, 2009

Department of Computer Science
Rhodes Univerity
South Africa

Acknowledgements

I acknowledge the financial and technical support of this project by Telkom SA, Comverse, Tellabs, Stortech, Mars Technologies, Amatole Telecommunication Services, Bright Ideas Project 39, THRIP and the NRF through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

Abstract

Introducing firewalls and other choke point controls in existing networks is often problematic, because in the majority of cases there is already production traffic in place that cannot be interrupted. This often necessitates the time consuming manual analysis of network traffic in order to ensure that when a new system is installed, there is no disruption to legitimate flows.

An added complication that is often the case with legacies or other systems that have developed organically, is that documentation about existing legitimate communication may be very limited, or in some cases is incorrect. Furthermore ever increasing traffic volumes make manual traffic analysis less feasible. It is therefore desireable to automate as much of the firewall configuration process as possible.

To improve upon this situation a system facilitating network traffic analysis and firewall rule set generation was developed. A detailed overview of the design and implementation of such a system is presented and its functionality evaluated.

It is found that firewall rule set generation through passive traffic inspection is a positive approach that significantly reduces the time necessary to create firewall rule sets. However, user review and refinement of the generated rules is still required.

Contents

1 Introduction **1**
- 1.1 Problem Statement . . . 1
- 1.2 Specifications And Requirements . . . 1

2 Literature Survey **2**
- 2.1 Firewalls . . . 2
- 2.2 FirewallBuilder . . . 2
- 2.3 Qt . . . 2
- 2.4 NetFlow . . . 3
- 2.5 SQLite . . . 3
- 2.6 libpcap . . . 3

3 Design And Implementation **4**
- 3.1 Design . . . 4
  - 3.1.1 High Level Overview Of The System And Its Components 5
  - 3.1.2 The Traffic Analyser . . . 5
  - 3.1.3 Modes Of Operation And Commandline Flags . . . 8
  - 3.1.4 The Rule Generator . . . 8
  - 3.1.5 The Pipeline . . . 9
  - 3.1.6 Code Names For The Componets Of The Project . . . 10
- 3.2 Implementation . . . 10
  - 3.2.1 The traffic analyser . . . 10
  - 3.2.2 The Rule Generator . . . 15

4 Testing And Results **18**
- 4.1 Testing . . . 18
  - 4.1.1 Performance Testing Of The Traffic Analyser . . . 18
  - 4.1.2 Network Firewall Configuration Case Study . . . 21
- 4.2 Results . . . 30
  - 4.2.1 Detecting SYN scans . . . 31

5 Conclusion **35**
- 5.1 Possible Future Extensions . . . 35

List of Figures

1 High level overview of the system and its position in the firewall configuration process 6
2 Class diagram of the traffic analyser 7
3 The Windows version of the visualisation script uses the sqlite3 executable in conjunction with common tools such as sed and awk to generate directed graphs in the dot language. 8
4 class diagram of the rule generator 9
5 One application of Charybdis' verbose mode is to visualise small trace file cross sections. Cross sections can easily be created by applying a *bpf* filter expression to a trace file with the analyser's -f or -F options. 11
6 The Charybdis help screen provides a reference to all but one undocumented command line flags. 12
7 The time taken for flow analysis increases proportioanally with the number of flows, whereas mode 2 analysis always performs linearly; a feature heavily used for performance evaluations during development. 13
8 Performance of the traffic analyser with different synchronous modi. Synchronous mode set to off outperformed the other modi by far. 13
9 The best performance in terms of speed is achieved by using a single transaction for all database accesses with snchronous set to off. 14
10 The Tetrix easter egg 16
11 Packet throughput speeds without logging to a database are significantly faster and positively identify the database as the biggest bottleneck within the system. 19
12 Without transactions the system is limited to very small traffic volumes, as disk IO is very slow. It is in this setup that the vast performance increase gained by disabling synchronous is most apparent. 20
13 Packet throughput with transactions increases performance in terms of speed significantly in all synchronous modi. 21
14 A new database opened in the rule generator 22
15 Investigating SMTP flows to find the networks MTAs. 23
16 SMTP traffic on the network visualised in a dot digraph.The machine in the centre is clearly the networks MTA. SMTP traffic contributions from external MTAs are represented by the number of arrows pointing towards it. 24
17 Removing an IP protocol in Scylla. 25
18 The TCP services tab allows to delete or insert TCP flows into the system. 25
19 The UDP services tab allows to delete or insert UDP services into the system. 26

20 The ICMP tab allows to delete or insert ICMP types and codes. 26
21 The IP tab defines what address range should be considered "inside" the firewall. 27
22 Excerpt of a FirewallBuilder network object file created by Scylla. 27
23 The exported rules opened in FirewallBuilder 28
24 Selecting a target firewall solution. 29
25 Compilation successful! 29
26 Excerpt from an ipfw rule set created for FreeBSD 30
27 A SYN scan on 196.23/16 32
28 Part of the same SYN scan as in figure 27, but this time visualised on 196/8. 33

1 Introduction

Focus of the project was to create a system capable of automatically generating a set of firewall rules by inspecting the underlying network traffic at a proposed choke point. Why the development of such a system is desireable shall be elaborated in the following subsection and the project specifications and requirements thereafter.

1.1 Problem Statement

Introducing firewalls and other choke point controls in existing networks is often problematic, because in the majority of cases there is already production traffic in place that cannot be interupted. This often necessitates the time consuming manual analysis of network traffic in order to ensure that when a new system is installed, there is no disruption to legitimate flows.
An added complication that is often the case with legacies or other systems that have developed organically, is that documentation about existing legitimate communication may be very limited, or in some cases is incorrect.
Furthermore ever increasing traffic volumes make manual traffic analysis less feasible. It is therefore desireable to automate as much of the firewall configuration process as possible.

1.2 Specifications And Requirements

A working system or series of tools that facilitate(s) the analysis of either live traffic or a recorded *pcap* trace file is to be developed. The output of this system should be useable as input for a tool such as *FirewallBuilder*[1], which allows for cross platform deployment on a wide variety of firewalling solutions. Ideally the resultant tool chain should be cross platform and be able to run on Unix-like and Windows systems. Main target systems and their corresponding firewall solutions are *ipfw* and *pf* on *FreeBSD*, *iptables* and *ipchains* on *Linux*, firewall policies on *Windows* and *ACL* on Cisco.

A user should be able to select from matching legitimate traffic flows at the following sorts of granularity: IP and IP level communications, Protocol level such as TCP, UDP etc., Port and Service level and a facility that handles ICMP types and codes should be provided.

The remainder of the paper is structured as follows: section 2 provides background information on the concepts, technologies and building blocks involved in the creation of the system. Section 3.1 presents a high level overview of the design of the system and explains where it fits into the firewall configuration process. Section 3.2 details the steps taken to implement the system. Section 4.1 presents the methodologies employed to test the system and the results of these test are then detailed in section 4.2. Section 5 concludes the paper and subsection 5.1 briefly outlines possible future extensions.

2 Literature Survey

This chapter serves to provide a brief overview of the concepts, technologies and libraries used to implement the project. The section on *NetFlow* was included to illustrate the concept of a traffic flow and why this representation is advantageous when dealing with network traffic, even though the project makes no direct use of *NetFlow*.

2.1 Firewalls

A firewall lends its name from the architectural structure of the same name, which is a specialised wall structure that prevents fires from spreading from one part of a house to another. In the context of computer networking a firewall is any hardware and or software device that prevents unwanted network connections to be established to or carried over a network. Firewalls are categorised according to their design, purpose and scope. There are host based firewalls, protecting the single host they are running on, dedicated firewalls and dedicated firewall appliances. In reality a firewall is rarely a single part of hard- or software but a combination of the former to varying degrees[18, 14]. All firewalls perform packet filtering of some sort or another, by selectively allowing only certain specified kinds of traffic through.

2.2 FirewallBuilder

FirewallBuilder is a GUI-based firewall configuration and management tool that supports *iptables* (netfilter), *ipfilter*, *pf*, *ipfw*, *Cisco PIX* (FWSM, ASA) and *Cisco routers extended access lists* [1]. It features a set of policy compilers that compile the rule sets created from within its GUI, from xml based object files, into, firewalling solution specific, firewall rule sets. The policy compilers do also create automatic deployment scripts, that allow the firewall to be brought up remotely and to roll back the installation if necessary. *FirewallBuilder* also ensures that the SSH connection between the configuring host and the firewall will never accidentally be interrupted. Because *FirewallBuilder*'s GUI is built upon *Qt* [2, 10] it is capable of running on a wide variety of target platforms, such as *Linux*, *FreeBSD*, *OpenBSD*, *Mac OS X* and *Windows* [5]. All of the above mentioned features make *FirewallBuilder* the ideal backend for the project.

2.3 Qt

Qt is a cross-platform application development toolkit for C++[2, 10]. It is used in a wide variety of software, most notably in the KDE, Skype, Google Earth, the opera web browser and SUN's VirtualBox. *Qt* recommends itself for the project as *FirewallBuilder*'s GUI is itself based on *Qt*. But *Qt* is more than a mere widget set - it includes networking, SQL, XML parsing and threading capabilities that are all accessible through the same API on all its supported platforms, which greatly facilitates and increases portability.

Qt has been acquired by Nokia in 2008, which has subsequently LGPLed the windows version, which previously has only been available under a proprietary licensing scheme.

2.4 NetFlow

NetFlow is both a format and a technology. Initially developed in-house at Cisco, it has quickly become the *de facto* standard for network analysis and is used for a variety of purposes including, but not limited to, billing, network planning, traffic engineering and the detection and analysis of security incidents[8, 9]. *NetFlow* enabled devices can export flow data via a UDP based protocol to a *NetFlow collector*, which then files, filters and stores the flow data.

Flows are created by continuously analysing IP packets and categorizing them into IP flows. A packet is either categorized into an existing flow or creates a new flow. Finished or expired flows are then exported to the *NetFlow collector* via UDP. A flow is defined by seven key fields, namely, source IP, destination IP, source port, destination port, protocol, type of service and input interface. Any two packets sharing the same entries for all seven fields belong to the same flow [7]. There are several versions of *NetFlow*, some of which are more commonly used than others, namely versions 1, 5, 7, 8 and 9 that incrementally improve upon another and provide a richer feature set with more detailed flow records [8].

2.5 SQLite

SQLite[3] is a small in-process database solution that is highly portable accross platforms. It is serverless and requires no configuration. This allows to add database functionality to any program with minimal effort.

2.6 libpcap

libpcap[4] (*WinPcap*[6]) is a packet capture and analysis library that is almost universal across different platforms. It also includes an interface to the Berkeley Packet Filter, which allows to only consider certain types of traffic during packet capture.

The following chapter on design and implementation elaborates on how the described libraries were used in the project to create the system.

3 Design And Implementation

Fundamentally the project is an engineering problem. How can network traffic, consisting of individual packets and captured by *pcap*, be filtered and classified into flows and these flows in turn be exportet into a format understood by *FirewallBuilder* ?

Whatever system performes this task has to act as a middleware between these two endpoints, network traffic and *FirewallBuilder*, and thus makes automated firewall rule set generation possible. By inspecting traffic passively, the sytem basically becomes an input driven parser that, after a first pass, additionally accepts user input by allowing the user to select between the calssified flows created in pass one, which are then the basis for a second and final pass in which the *FirewallBuilder*-based precursors of the firewall rule sets are created. The following subsections on design and implementation elaborate on exactly how this was achieved.

3.1 Design

The system was initially envisioned in a classical client-server architecture, with the command line based traffic analyser acting as the server and the GUI-based rule generator acting as its client. However, during implementation focus shifted away from this approach as it became clear that no permanent two-way communication between the traffic analyser and the rule generator was actually required. Within a typical run of the system, information, in form of the traffic analysis database, is exchanged only once, from the traffic analyser to the rule generator. This insight was taken into account and consequently the system is now designed and implemented as two separate stand-alone applications and their information exchange is handled by traditional means of file exchange, such as FTP, SCP, HTTP or any other service that is capable of transmitting large files between hosts. While the traffic analyser encapsulates most of *tcpdump*'s functionality it isn't even necessary to install it on the proposed choke point if an analysis is only going to be performed once. *Tcpdump*, which is included in the default installation of most operating systems, can be used to create a *pcap* trace file which can then be transferred to an ideally powerful workstation to allow the traffic analysis and the consequent rule generation to take advantage of its underlying hardware.

If repeated runs of the traffic analyser are expected or analysis it to be performed on live traffic the traffic analyser can of course be deployed on the proposed choke point itself.

There is a distinct advantage to this modular approach of separating the analysis and rule genration functionalities. Because the traffic analyser is designed and implemented as a command line utility, it can be run remotely on machines without windowing capabilities, such as rack-mounted servers, while the GUI-based rule generator can take advantage of the convenience of a desktop machine.

The project places a strong emphasis on portability, an absolute necessity in today's inherently heterogenous networking environments.

In the following an overview of the two components and their position within the firewall configuration process shall be given.

3.1.1 High Level Overview Of The System And Its Components

As previously stated, the system acts as a middleware between the observed network traffic and *FirewallBuilder[1],* as is depicted in figure 1.

After the future choke point has been identified, its underlying network traffic is fed into the system. This input can be in either one of the following two formats. Firstly live traffic and secondly *pcap* trace files recorded at the node. These inputs are then processed by the traffic analyser which convertes them into a custom flow format that is largely inspired by *Cisco's NetFlow.* This format features various metrics such as source IP and port, destination IP and port, IP protocol, service level protocol, initiation time and packet length. The flows recorded by the traffic analyser are stored in a database. The rule generator then acts upon this database and allows the user to inspect, enquire about and modify the present flows on various levels of granularity such as IP, IP protocol, service, connection count and so forth. After this initial inspection by the user the rule generator proposes a set of rules that match the observed traffic and exports them into *FirewallBuilder* network object file format. The proposed rules can then consequently be reviewed and refined by the user from within *FirewallBuilde*r's UI, which in turn allows for the deployment and maintenance of the firewall.

3.1.2 The Traffic Analyser

The program entry point lies within the class charybdis. Charybdis creates itself one core which handles all functionality of the program. Charybdis is therefore somewhat like an outer shell that serves as a placeholder for future extensions. Core incorporates the global configuration file and spawns itself a PcapEngine and a SqliteEngine which handle packet capture and database access respectively. PcapEngine is linked against *libpcap* and incorporates the protocol definition file; it also handles protocol dissection and classification. SqliteEngine handles all database connection and access functions required by the analyser. A graphical presentation of this architecture is given below in figure 2.

The analyser uses *libpcap* [4, 13](*WinPcap [6]* on *Windows*) for the processing of live traffic and trace files. The handling of these two inputs within *pcap* is nearly identical. This, from a programmers perspective, very convenient API and its support for almost all major operating systems recommended *pcap* as the basic building block for the traffic analyser.

The strategy to obtain the information required for the custom flow format is to screen the packet data for three way handshakes and TCP FIN and RST packets.

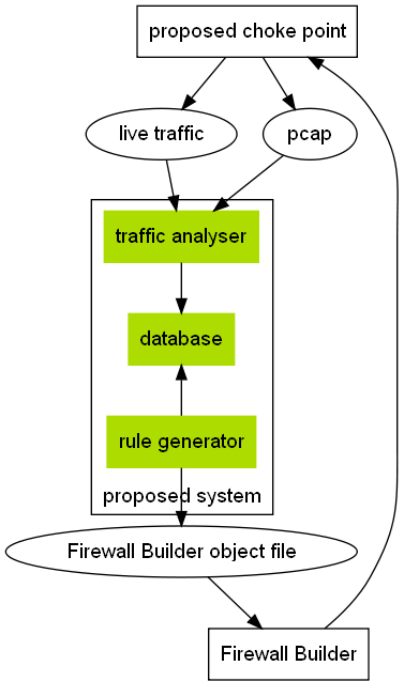

Figure 1: High level overview of the system and its position in the firewall configuration process

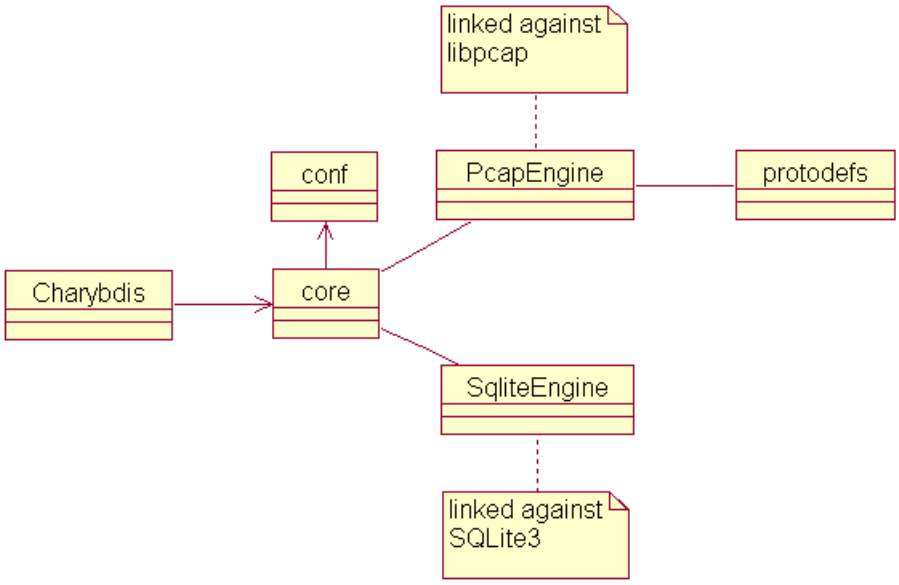

Figure 2: Class diagram of the traffic analyser

The ACK packets involved in the three way handshake can be determined through the packets' sequence numbers [16]. This establishes the sources and destinations and hence the direction of the traffic flows. The difference in the timestamps between these packets allows for an estimation of the duration of any given connection. The packets that are neither SYN nor FIN are matched to one of the existing flows and their payloads added to the total volume of traffic in the flow. Their occurrence is also recorded in the packet count of the flow. Because packets might arrive out of order, care must be taken when reconstructing the flows to not disregard valuable information, meaning that non SYN or FIN packets without a corresponding flow do create their own flow so that it is possible to reconstruct them later or at least take them into consideration when generating the rules at a later stage.

The project uses the embedded SQL database engine *SQLite* [3, 15] to record the flows, which has various advantages over other more sophisticated database solutions. From a performance perspective, this in-process library is simply much faster than any networked database solutions could ever be. Because *SQLite* is serverless, self-contained and requires no configuration it also increases the ease of use of the system. *SQLite* stores its databases in a file, in a format that is consistent across all platforms, thus its database files lend themselves as the perfect format for information exchange between the traffic analyser and the rule generator, especially across different platforms [15, 3]. There are two added

```
:: visWin.bat
:: queries a charybdis SQLite traffic database for src-dst flows
:: and pumps the output through fdp to create a directed dot graph
@echo off
sqlite3 -csv %1 "select sIP1,sIP2,sIP3,sIP4,dIP1,dIP2,dIP3,dIP4
    from %2" > visWinTmp.1
gawk -F, "{print \"*\" $1 \".\" $2 \".\" $3 \".\" $4 \"* -# *\" $5
    \".\" $6 \".\" $7 \".\" $8 \"*\"}" visWinTmp.1 > visWinTmp.2
sed -e s/#/">"/g visWinTmp.2 > visWinTmp.3
sed -e s/*/\"/g visWinTmp.3 \" > visWinTmp.4
echo digraph G { > visWinLayout.dot
cat visWinTmp.4 >> visWinLayout.dot
echo } >> visWinLayout.dot
rm visWinTmp.*
fdp -Tjpg -o visWinLayout.jpg -Kfdp visWinLayout.dot
visWinLayout.jpg
```

Figure 3: The Windows version of the visualisation script uses the sqlite3 executable in conjunction with common tools such as sed and awk to generate directed graphs in the dot language.

advantages to this approach, firstly database files can be efficiently compressed and are therefore ideal to be sent across the network and secondly this 'offline' recording of flows does not create any network traffic associated with the recoding process itself, that would otherwise create artefacts and would have to be filtered out.

3.1.3 Modes Of Operation And Commandline Flags

By default the traffic analyser exhibits a very low level of verbosity, this is largely due to the fact that outputting characters onto a terminal is a relatively slow process on almost all operating systems. Some sort of visual feedback about the state of the program is however desirable and increases the user experience. To accomodate human psychology visual feedback is provided by a spinner that indicates whether or not packets are being processed. If the verbosity flag is set the program will output its current state and the flow directions of the packets currently analysed onto the terminal. This can be useful for debugging purposes or when working with small trace files or short periods of live capture. It is also a valuable visualisation tool when working with *pcap* cross sections on remote terminals where the repeated transmission of visualisation script output might be undesirable.

3.1.4 The Rule Generator

The rule generator code named Scylla, is basically a frontend to the traffic and flow database files created by Charybdis. It is GUI-based and built upon the *Qt* library. *Qt* was chosen because of its extraordinary portability and homogenity accross platforms. The fact that *Qt* is a superset of C++ also

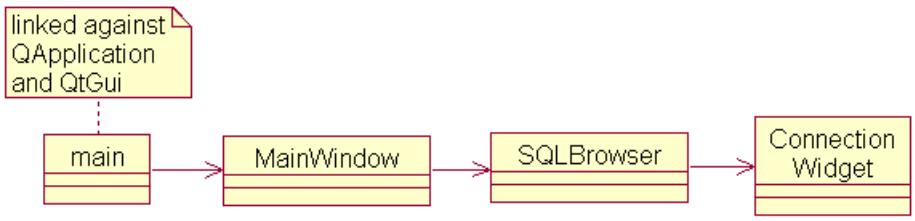

Figure 4: class diagram of the rule generator

facilitated development as it allowed the entire project to be implemented in this language.

From an architecureal point of view, the design of the rule generator is very straight forward. A main class instantiates the MainWindow class which contains the window frame and all menues, tool-, and statusbars. The central widget of the MainWindow is the SQLBrowser which contains all tabs and actions that carry database functionality. The actual database connection is created and maintained through SQLBrowser's ConnectionWidget class, which contains the database drivers and handles low level database access as is shown in figure 4.

The traffic databases can be acted upon from within the rule generator using the Structured Query Language, which might seem a little spartanic at first but is certainly less cryptic than Berkeley Packet Filter expressions with which the average user is certainly less familiar. Just like it is often argued that user friendly and highly abstracted systems take a performance hit because of the complexitites of the abstraction mechanisms, the author would like to argue that the same is true for usability vs. versatility. While the mainly SQL based interface might be less friendly towards the average computer user it definitely allows for a versatility that cannot be parralled by any graphical representation of traffic flows.

3.1.5 The Pipeline

The architecure of the project as was outlined above basically forms a filtering pipeline with various stages that can all be employed to filter the recorded traffic down into humanly manageable chunks. The first filter is *pcap,* and associated with it *bpf,* which can be harnessed by Charybdis' -f and -F options, which allow for a very fine grained control of what types of traffic are even considered by the system. The second equally powerful filter is the database and its associated SQL interface from within the rule generator's GUI. Scripts to prune or otherwise act on the database can be deployed from both the rule generator and traffic analyser alike. The ultimately last stage of the processing pipeline is *FirewallBuilder* itself, in which all classified flows are disabled by

default and can be selectively enabled by ther user.

All but the last stage, *FirewallBuilder*, have applications beyond the initial objective of the project.

The traffic analyser and its associated filters could for example be used as a reconnaisance tool for penetration testers. An immense amount can be learned about a network without sending out a single packet and thus minimizing the danger of detection by an IDS. Please refer to section 4.2.1 for an example of how to use the system to detect SYN scans.

3.1.6 Code Names For The Componets Of The Project

The code names for the rule generator and the traffic analyser are Scylla and Charybdis respectivly. The well know phrase "between Scylla and Charybdis" has its origin in ancient greek mythology and refers to a situation in which one has to choose between two options which are both guaranteed to have catastrophic effects. Scylla and Charybdis, two sea monsters guarding a narrow strait ment almost certain demise to all sailors trying to pass through it. Charybdis was a gigantic mouth in the sea which would, at random intervals, suck in water and ships alike, a more than fitting metaphor for a traffic analyser.

Scylla on the other hand was, according to Ovid, a once beautiful nymph and love interest of the fisherman turned sea god Glaucus. When she turned him down and escaped onto land where Glaucus could not follow, he sought the help of the witch Circe, which in turn fell in love with Glaucus. Angry with Scylla for turning Glaucus down, Circe cursed Scylla and turned her into a six-headed monster. In Homer's Ilias Ulysses is faced with the task of crossing between Scylla and Charybdis and chooses to pass closer to Scylla, a decision that costs him six of his men. The metaphor for the rule generator being the selection of classified flows instead of sailors.

3.2 Implementation

This section describes the implementation of the componets that make up the system. It also highlights the roles of the libraries and toolkits upon which the system was implemented, individually. For the libraries, a brief justification as to why they were chosen shall be given as well. A major driving force behind the selection of the libraries and toolkits is their portability and licensing. A strong emphasis is put on the use of free and open source software since high portability is a fundamental requirement in the highly heterogenous environment of todays networking infrastructures.

3.2.1 The traffic analyser

The traffic analyser, code named Charybdis, is capable of detecting all 142 protocols that IANA currently endorses as being transportable over IP. Furthermore its protocol dissectors recognize 138 services transported over TCP, 106 services transported over UDP and all but the reserved ICMP types and all

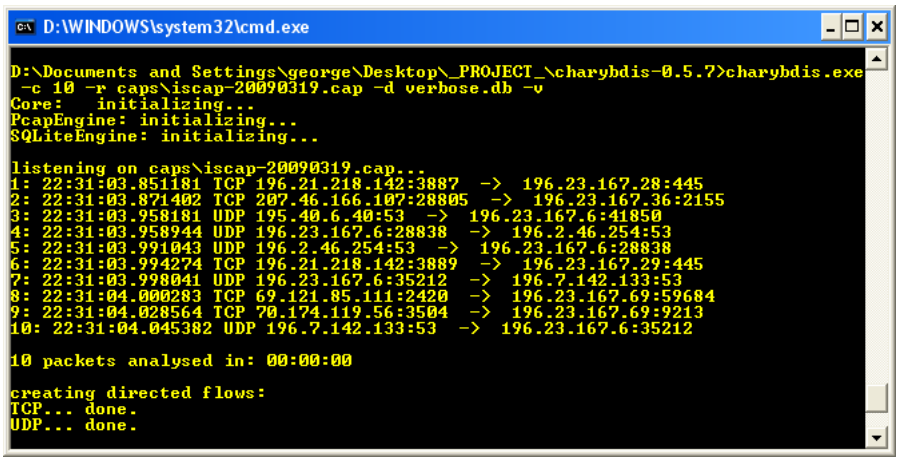

Figure 5: One application of Charybdis' verbose mode is to visualise small trace file cross sections. Cross sections can easily be created by applying a *bpf* filter expression to a trace file with the analyser's -f or -F options.

of their respective codes. All packet capturing is performed by utilising the *pcap* library, this applies to both live traffic and *pcap* trace files alike. The following paragraph provides a short introduction to the traffic analyser's purpose and its functionality.

The traffic analyser's primary task is to create or extract traffic flows from its input data and to consequently store these flows in a database. Working with flows is advantageous because of their high information density[17, 11, 12], and because they contain stateful information about the prevalent network connections.

Because the traffic analyser relies heavily on *pcap* and *SQLite*, which are both implemented in C/C++ and provide native interfaces to their functionality in that language, its use for the implementation of this part of the project was an easy choice. Implementing the traffic analyser in C++ has other advantages which are mainly the high performance of natively compiled executables and the relative ease of porting between different platforms. C++ programs are often considered to be hard to port, but conditional compilation and the wide availability of C++ compilers accross different platforms, especially the *GCC* has made porting quite feasible within the scope of the project. On Windows the *MinGW* version of *GCC* that ships with *Qt* was used to compile the project, traffic analyser and rule generator alike. Portability was further enhanced by the fact that the componets upon which the traffic analyser builds are already available on all targeted platforms. *SQLite* and *pcap* are both almost universal to Windows, Linux and the *BSDs.

The biggest hurdle that had to be taken while implementing the traffic analyser was the apparently slow speed of SQLite. This however turned out to be a well known problem and can be overcome by firstly tuning SQLite by setting

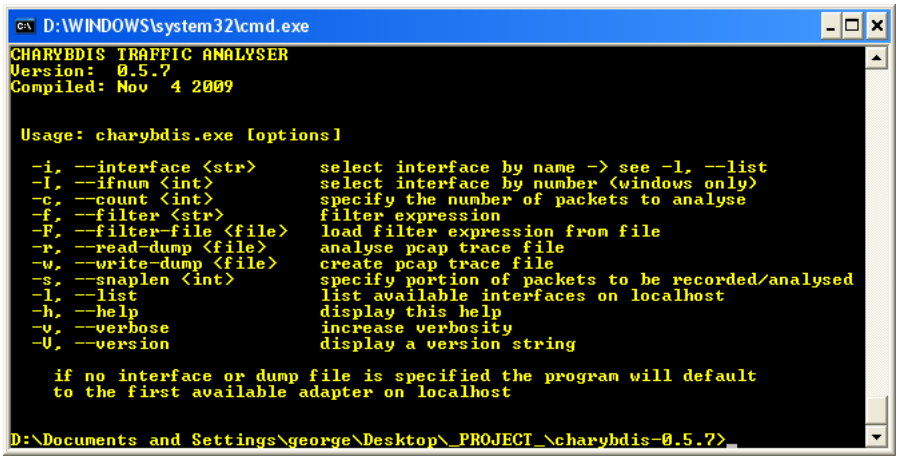

Figure 6: The Charybdis help screen provides a reference to all but one undocumented command line flags.

the synchronous pragma to off and secondly by conduction all database accesses within a single transaction.

The *SQLite* documentation states that it is capable of up to 50000 inserts per second, however, insertion speeds of this magnitude are only achieved if the synchromous pragma is set to off, which instructs *SQLite* to continue operation as soon as data was handed over to the operating system. While this mode of operation can be up to 50 times faster, it comes with the drawback that a powercut or application crash might leave the database corrupted as not all information might have been written to disk, yet.

In the context of the traffic analyser there is simply no other choice than to set the synchronous pragma to off because operation in synchronous modi normal and full are simply prohibitivly slow as is illustrated in figure 8.

As is shown in figure 8 the speeds achieved by disabling the synchronous mode are still not nearly fast enough to record any reasonable amount of traffic let alone high traffic volumes, and further optimizations were necessary. The breaktrough in terms of database speed was achieved when the entire traffic analysis was treated as a single database transaction, meaning that all inserts and table creation processes were enclosed by a single BEGIN - COMMIT statement, as is illustrated in figure 9.

While the overall speed up is significant, the transaction approach seems to diminish the speed up contribution of the synchronous setting. Why this might seem to be the case is further explored in the results section.

Measurements with the undocumented mode 2 in which no flow analysis is performed and packets are indiscriminately added to a packet database show a 171 fold speed improvement of transaction/off over without transaction/normal.

The initial flow analysis is performed after all the packets have been added to the database and the first pass of the traffic analysis has come to an end.

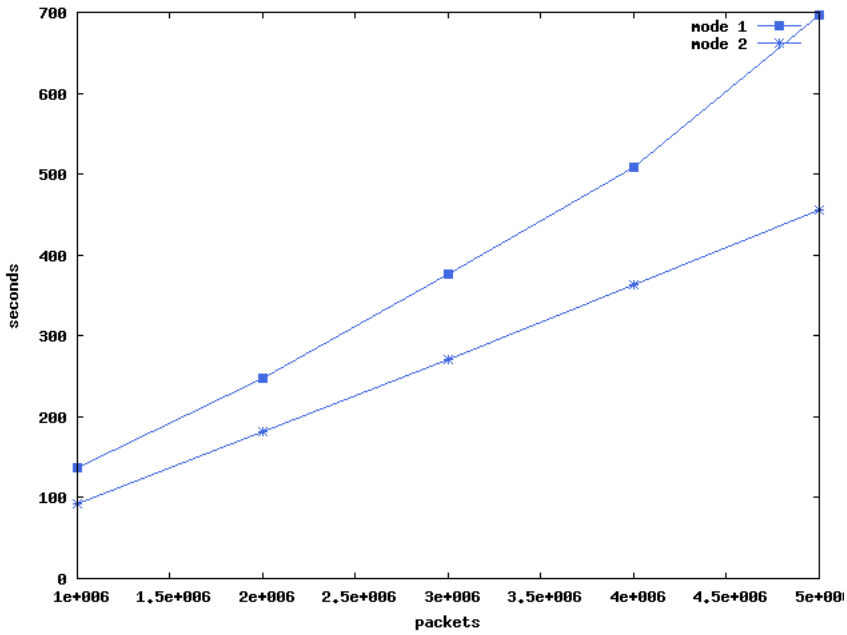

Figure 7: The time taken for flow analysis increases proportioanally with the number of flows, whereas mode 2 analysis always performs linearly; a feature heavily used for performance evaluations during development.

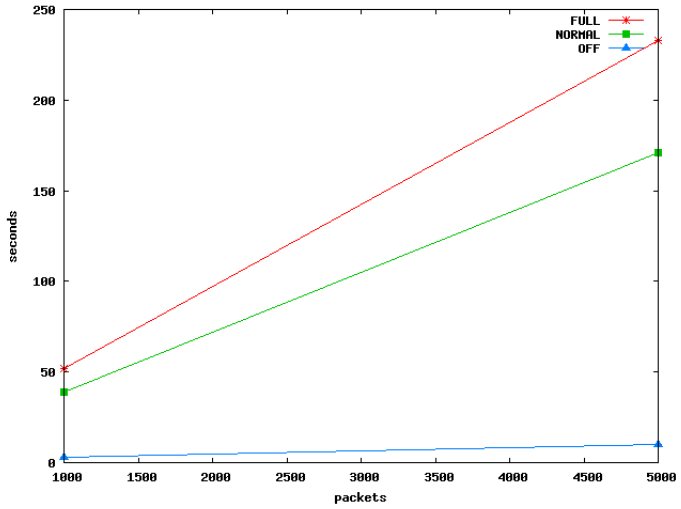

Figure 8: Performance of the traffic analyser with different synchronous modi. Synchronous mode set to off outperformed the other modi by far.

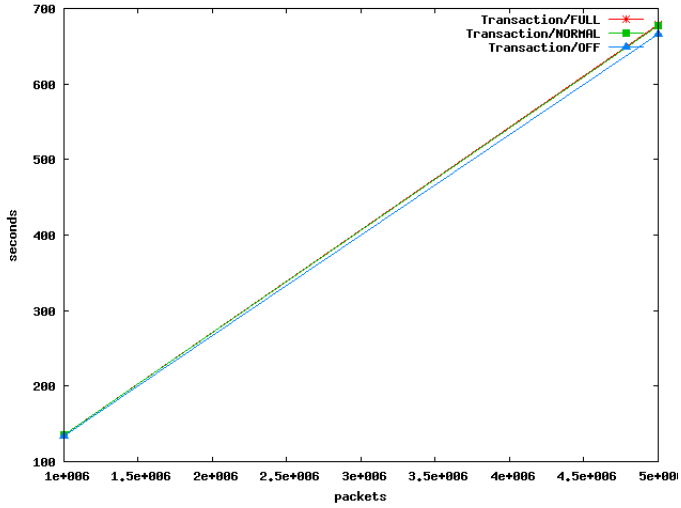

Figure 9: The best performance in terms of speed is achieved by using a single transaction for all database accesses with snchronous set to off.

This is for the technical reason, that even though *SQLite* is perfectly capable of 50000 inserts per second, updates are very much slower and the time needed to find a certain flow in order to update its state increases with the number of flow records already added to the database. This is illustrated in Figure 7. Flow records are initally created in a format that is a stripped down version of *NetFlow* and carry no stateful flow information by default, since stateful connection details are not regarded in *FirewallBuilder* and adding state information to the flows in *FirewallBuilder* is much easier and quicker than extracting this information from the flow database.

In a default mode 1 analysis the traffic anlayser creates a set of tables, that each fullfil a specialised purpose. Some of the tables are highlighted in the following: The FRAMES table containes all observed packets individually and is the basis for all consecutive steps of the flow analysis. It is the only table that is also created in mode 2. From this FRAMES table one table is create for every IP protocol identified during the analysis. These protocol tables have the extension _raw. From each of these tables another table with just the name of the protocol without the _raw extension is created, in which only distinct flows between hosts are recorded. This is an absolute necessity to prune the tables to a humanly manageable size. As a final step a table of all found protocol names is created that is the basis for the IP protocol pruning tab within the rule generator. Should the actual protocol flow tables ever be corrupted they can easily be reconstructed from the _raw tables, that are left untouched in the database for this very reason.

The traffic analyser also features the previously mentioned undocumented mode 2 in which no initial flow analysis is conducted. This feature is a leftover

from a debugging feature used during development, but was intenionally left intact as it can be harnessed by the experienced user not only to speed up the initial flow analysis by disregarding certain IP protocols, like UDP which takes by far the longest to analyse, but also to allow to create flows based on custom filters. Mode 2 was used extensively for packet throughput timings during develoment and during this wite up.

3.2.2 The Rule Generator

The rule generator uses the database file created by the ananlyser to propose firewall rule sets that match the observed traffic. The different types of flows recorded in the database are each individually visualized in a table view where they can be closely inspected and modified by the user. A facility to perform custom SQL queries against the database is also provided within the GUI and is a very prominent and powerful aspect of the system. Within the connection widget the table structure and field names of all tables can be enquired, by right clicking on a table and selecting 'show metadata'. All functions within the GUI can also be invoked by means of keyboard shortcuts to create a highly responsive working environment. This allows SQL queries to be issued in quick sucession, so that the thought process of querying the database is uninterupted by repeated point and click actions. The output of all custom queries can be sent to the scripting interface. The scripting interface itself is actually a functionality of the traffic analyser, but it can also be conveniently accessed from within the rule generator. One application for the scripting interface is to investigate certain kinds of flows accross the network. If for example an exotic service like the Andrew File System is found on the network the user can create a table that holds all connections of this type by issuing a command like:

```
create table AFS as select * from UDP where service like 'AFS%'
```

This will pull all AFS services (there are at least seven distinct ones, all running on different ports) into a single table which then in turn can be visualised by invoking the visualisation script. An example of how to use this technique to find for example the MTA of a network is given in the section on testing. How powerful this visualisation capability is cannot be overstated. Understanding raw IP addresses or even DNS names and keeping the relevant enpoints and directions of the involved flows in mind is certainly much harder than having the flows of interest represented graphically.

At the end of the inspection, review and refinement process the user is able to export the policies into a *FirewallBuilder* network object file. The basic strategy to automatically generate a rule set is to divide the network into an 'inside the wall' and an 'outside the wall' part. All flows carried between inside and outside are split into an inbound and an outbound flow and the corresponding policies are created. Initially all flows are denied.

In order to export to *FirewallBuilder*, its network object file format had to be reverse engineered. This was achieved by employing a black box approach. A sample file was created and an object such as a policy, a group, an address

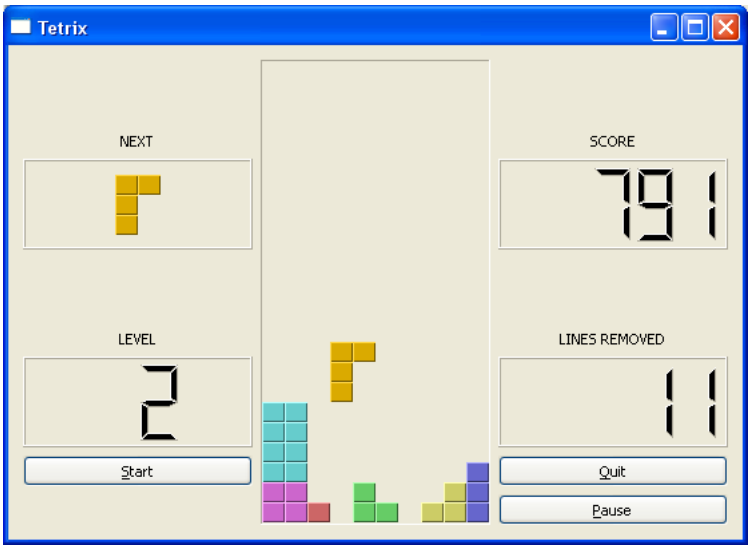

Figure 10: The Tetrix easter egg

or a network service added to it. The resulting changes were then observed in the XML based network object file itself. The attributes for most elements are very verbose and self explanatory and presented no problem to replicate. The id attribute of all elements turned out to be more cryptic but definitely follows a pattern. A random 4 digit number followed by the character X, followed by a three digit number. The number after the X remains static within all subelements of any given element whereas the number before the X is continuously incremented for every new subelement within an element. Because it is unclear whether these numbers are random or carry a deeper meaning the numbers created by the template were replicated and the plus one algorithm applied wherever id tags have to be generated dynamically. Because elements frequently reference earlier elements, all id tag ranges for every element are temporarily stored in integer variables. No library or XML functionality within Qt was used to create the XML file, but plain printf statements that insert the dynamic parts of the network object files by means of simple loops.

When working with large datasets, as the program is intended to do, or when issuing complicated queries or calling scripts on large datasets, that take a long time to return one has the option of passing the time with the hidden Tetrix easter egg, that is depicted in figure 10. The keyboard shortcut to start a new instance of Tetrix is [CTRL+T].

In conclusion it can be said that the design and implementation of the project was successful and that the resulting system is not only functioning to specification but also demonstrates capabilities beyond the initial scope of the project. It was also outlined how a few technical hurdles that presented themself during implementation were incrementally overcome.

The following chapter four details how the implementation was stress tested, on what what hard- and software platforms these tests were conducted and how the system performed in these test.

4 Testing And Results

The two components of the system, traffic analyser and rule generator, were initially tested separately to evaluate their relative performance within their respective problem domain. Focus of the analyser tests was to identify and eradicate bottlenecks as much as possible to tune performance in terms of speed. The rule generator was only indirectly tested, since it fullfils more the function of a front-end and relies heavily on the traffic analyser. The rule generator's rule export function is known to be working correctly, since incorrect *FirewallBuilder* network object files simply will not be opened by *FirewallBuilder*. Rules are exported in the file format of *FirewallBuilder* 3.0.5, previous versions are not supported because subsequent releases of FirewallBuilder add new attributes to the file format which are not understood by previous versions.

4.1 Testing

The system was thoroughly tested to ensure that it can deal with loads of the required magnitude.

All testing was performed on a machine with the following hardware and software specifications:

```
Intel Pentium 4 at 3 GHz
1024 MB RAM.
SATA diskcontroller / disks
Windows XP Professional SP2
```

Testing of the analyser was focused on speed optimizations, whereas the main focus for testing of the rule generator lay within the domain of correct rule generation.

4.1.1 Performance Testing Of The Traffic Analyser

In most situations the actual time taken for a traffic analysis is not the main concern for the usability of the system, since its operation is unattended. However it has to be within workable limits, i.e. an analysis that takes longer than the period of observed traffic that is used as its input is certainly undesirable. Therefore the analyser was sped up as much as possible. Focus of all tuning activities was the SQLite interface as its disk IO is certainly the main bottleneck, as was established by simply testing analysis speed with and without the –database option. The results of this test are conclusive and illustrated in figure 11.

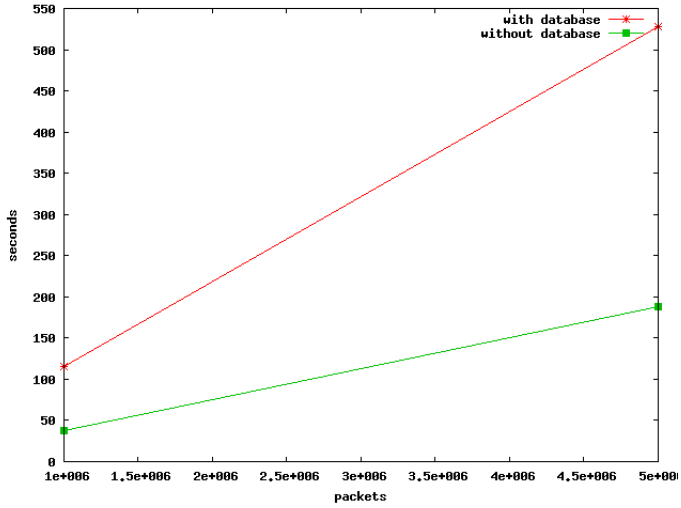

Figure 11: Packet throughput speeds without logging to a database are significantly faster and positively identify the database as the biggest bottleneck within the system.

Timings were taken in mode 2, that measures only packet throughput time. In the analysers default mode with transactions enabled and synchronous set to off, 1000000 packets are processed in 115 seconds and 5000000 packets are processed in 528 seconds.When packets are not recorded in a database, throughput times for 1000000 and 5000000 packets decrease to 37 and 118 seconds respectively.

When working with a database, as is the case during normal operation, vast performance increases in terms of speed can be achieved by setting the synchronous pragma to off and by enclosing all database accesses in a single transaction.

The aforementioned undocumented mode 2, in which no initial flow analysis is conducted, again served as a basis for the speed up evaluations of these two settings. Conveniently for throughput testing, mode 2 database access times increase linearly with anaysed traffic volume. The incremental performance increases obtained by applying the two previously stated tweaks to the system are documented in the following.

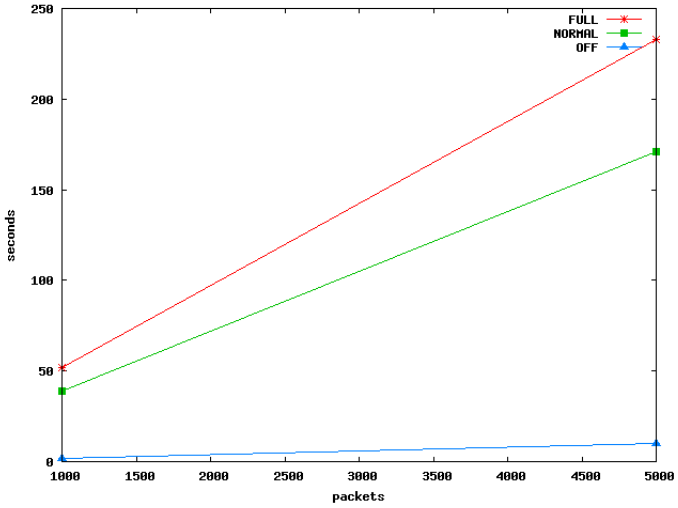

Figure 12: Without transactions the system is limited to very small traffic volumes, as disk IO is very slow. It is in this setup that the vast performance increase gained by disabling synchronous is most apparent.

The performance gain of synchronous mode set to off over full synchronous mode without transactions is 26 fold as is depicted in figure 12.

If all database accesses are enclosed in a single transaction the performance increase in terms of speed gained by disabling synchronous is only realised once, as all operations on the database are treates as one atomic operation. This is illustrated in figure 13.

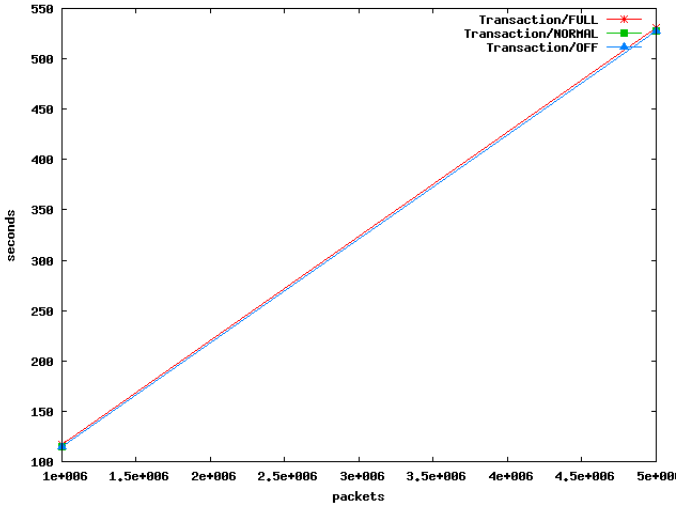

Figure 13: Packet throughput with transactions increases performance in terms of speed significantly in all synchronous modi.

While it might seem that the employed synchronous mode loses its significance if transactions are enabled, this is certainly not true. Obviously the above timings were recorded in mode 2, and the speed ups gained by the faster disk access only begin to show in the normal mode of operation, mode 1. In mode 2 5000000 packets are recorded in the database in 531 seconds if synchronous is set to full, but in only 528 seconds if synchronous is set to off. In mode 1, in which the number of database inserts is significantly higher, the speed up gained by using synchronous off shows very clearly. 5000000 packets are processed a full 19 seconds faster when synchronous is set to off. This effect is accumulative when working with dataset very much larger than 5000000 packets.

The second bottleneck in the operation of the traffic analyser is verbosity as print to screen operations are relatively slow on most operating systems. The design decision to keep verbosity to a minimum by default is definitely rewarded by the observed speed ups. A 1000000 million packet analysis in mode 2 takes 1 minute 58 seconds whereas the same operation with the verbosity flag set completes only after 7 minutes and 14 seconds.

4.1.2 Network Firewall Configuration Case Study

The functionality of the system was tested using *pcap* trace files recorded at one of the schools in Grahamstown. Objective was to create a firewall for the class C network segment 196.23.167/24. The 4.28 GB trace file contained just over 47 million packets which accounted for one week's worth of traffic. On a file of this size the total analysis time was 5 hours 46 minutes and 22 seconds of which only 1 hour 39 minutes and 59 seconds were needed to add the traffic into the database, while the remaining 4 hours 6 minutes and 23 seconds were taken to

complete the initial flow analysis.

After Charybdis' analysis run completed, the resulting flow database could be inspected from within Scylla, as is depicted in Figure 14.

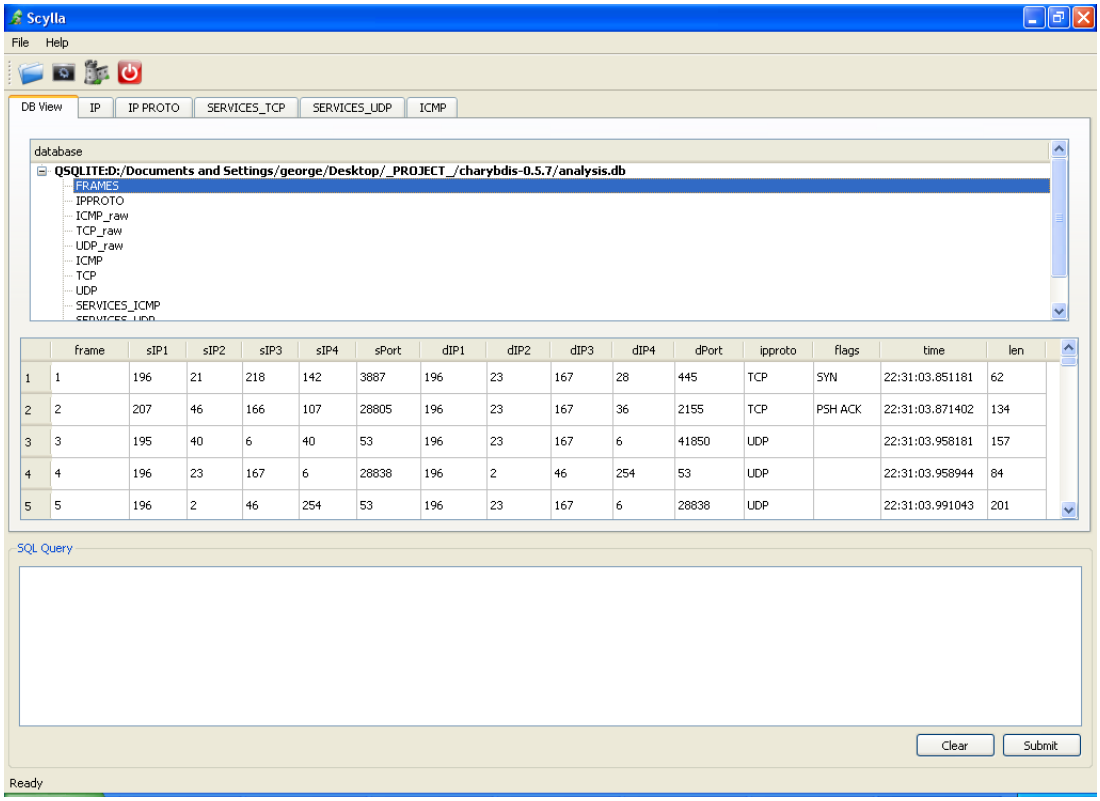

Figure 14: A new database opened in the rule generator

At this stage it is now possible to create custom flows to investigate the network layout as is shown in Figure 15.

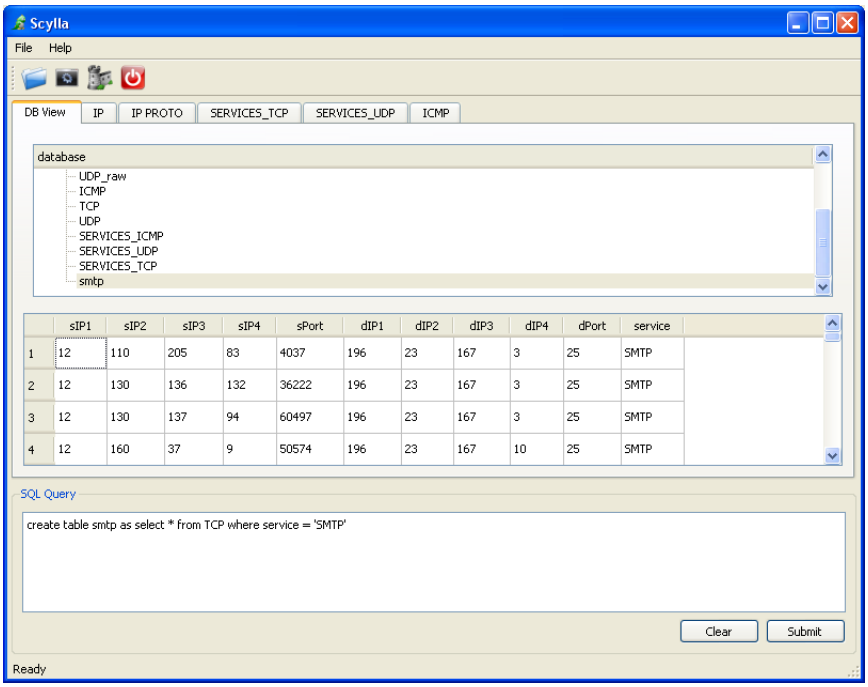

Figure 15: Investigating SMTP flows to find the networks MTAs.

With the help of the visualisation script networking infrastructure like mail servers can be identified very quickly, as is shown in figure 16.

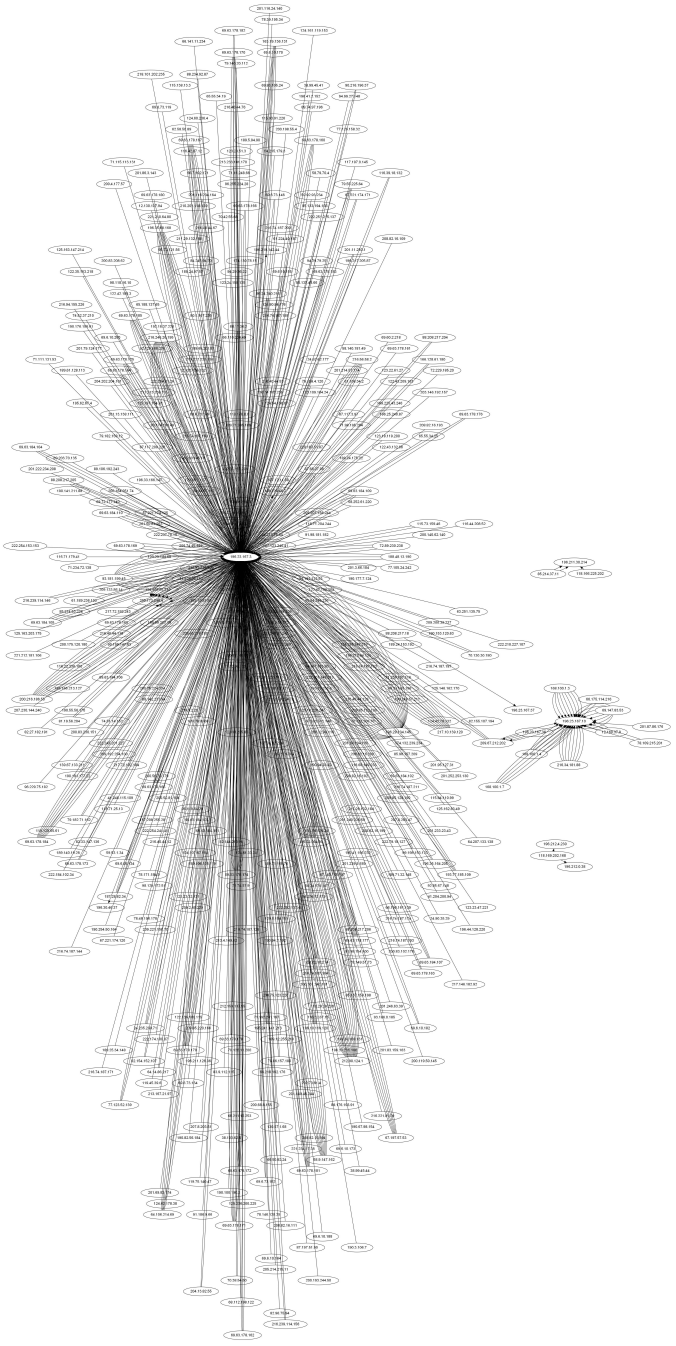

Figure 16: SMTP traffic on the network visualised in a dot digraph.The machine in the centre is clearly the networks MTA. SMTP traffic contributions from external MTAs are represented by the number of arrows pointing towards it.

After an initial understanding of the network and its key services and flows is attained, the user can now make informed decisions as to what kind of traffic should be allowed to be carried on the network. All subsequent tabs allow to delete or insert protocols and services.

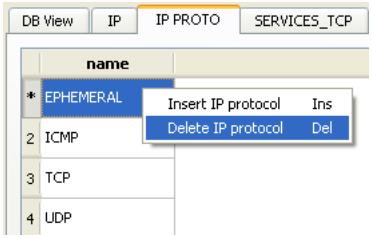

Figure 17: Removing an IP protocol in Scylla.

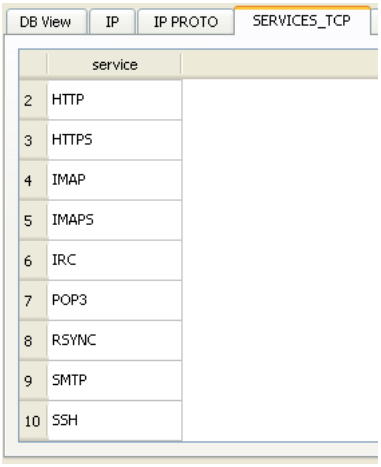

Figure 18: The TCP services tab allows to delete or insert TCP flows into the system.

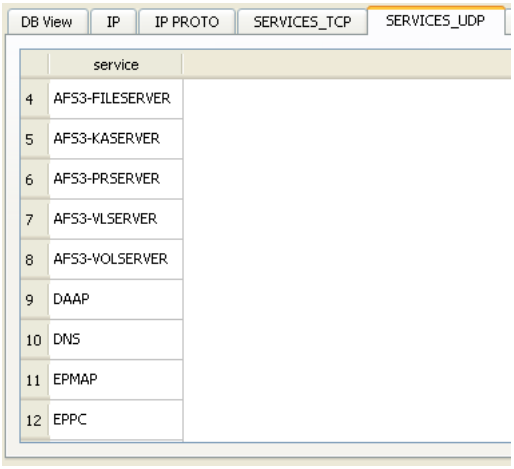

Figure 19: The UDP services tab allows to delete or insert UDP services into the system.

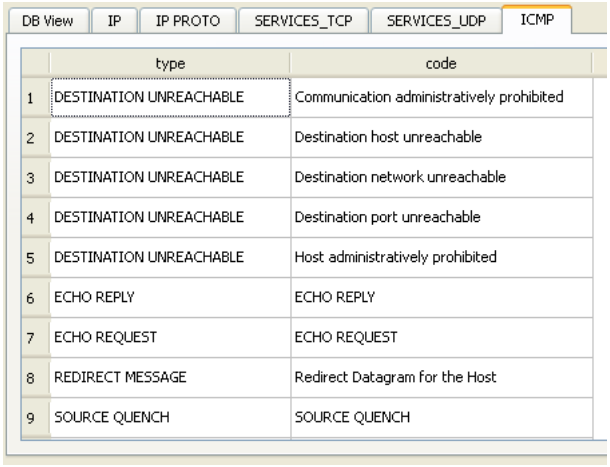

Figure 20: The ICMP tab allows to delete or insert ICMP types and codes.

Pruning the present flows this way is a very quick and easy way to disregard entire protocol suites, it is however not necessary to do all flow selection from within Scylla, as after the flows have been imported into *FirewallBuilder* the user will have another chance to select between classified flows that should be allowed on the network on an even finer grained level.

Before the flows are exported to *FirewallBuilder* network object file format, the part of the network that is to be protected by the firewall is specifyed in the IP tab, as can be seen in figure 21.

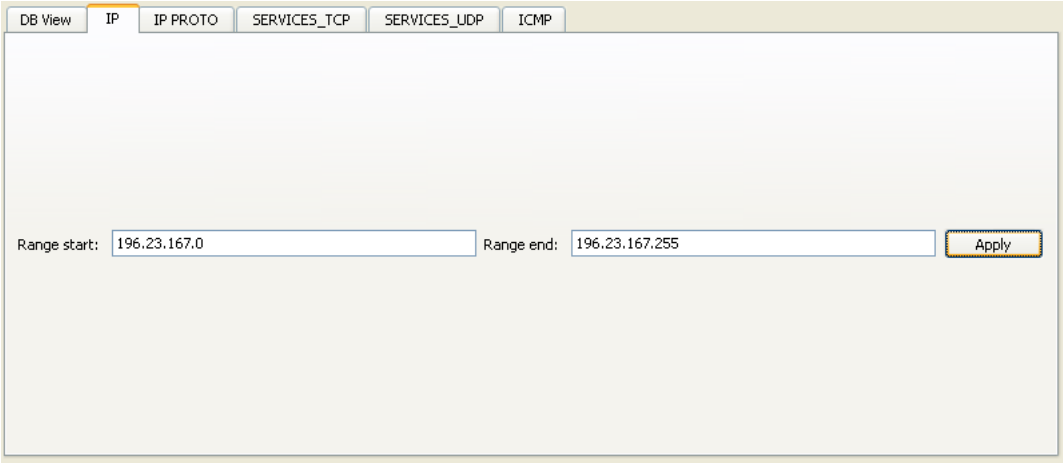

Figure 21: The IP tab defines what address range should be considered "inside" the firewall.

With a click on the *FirewallBuilder* icon in the toolbar or by invoking the shortcut [CTRL+E] the user is promted to which file the *FirewallBuilder* object file should be saved to. An excerpt of such an XML file is shown in figure 22.

```
        <ObjectRef ref="id1958X172"/>
        <ObjectRef ref="id1959X172"/>
        <ObjectRef ref="id1960X172"/>
        <ObjectRef ref="id1961X172"/>
        <ObjectRef ref="id1962X172"/>
        <ObjectRef ref="id1963X172"/>
        <ObjectRef ref="id1964X172"/>
        <ObjectRef ref="id1965X172"/>
        <ObjectRef ref="id1966X172"/>
        <ObjectRef ref="id1967X172"/>
        <ObjectRef ref="id1968X172"/>
        <ObjectRef ref="id1969X172"/>
        <ObjectRef ref="id1970X172"/>
        <ObjectRef ref="id1971X172"/>
        <ObjectRef ref="id1972X172"/>
        <ObjectRef ref="id1973X172"/>
        <ObjectRef ref="id1974X172"/>
        <ObjectRef ref="id1975X172"/>
      </ObjectGroup>
    </ObjectGroup>
    <ObjectGroup id="id3188X372" name="Hosts" comment="" ro="False"/>
    <ObjectGroup id="id3189X372" name="Networks" comment="" ro="False"/>
    <ObjectGroup id="id3190X372" name="Address Ranges" comment="" ro="False"/>
  </ObjectGroup>
  <ServiceGroup id="id3191X372" name="Services" comment="" ro="False">
    <ServiceGroup id="id3192X372" name="Groups" comment="" ro="False"/>
    <ServiceGroup id="id3193X372" name="ICMP" comment="" ro="False"/>
    <ServiceGroup id="id3194X372" name="IP" comment="" ro="False"/>
    <ServiceGroup id="id3195X372" name="TCP" comment="" ro="False">
      <TCPService id="id6384X1868" ack_flag="False" ack_flag_mask="False" established="False" fin_flag="False"
                  psh_flag_mask="False" rst_flag="False" rst_flag_mask="False" syn_flag="False" syn_flag_mask="
                  urg_flag_mask="False" name="DNS" comment="" ro="False" src_range_start="0" src_range_end="0"
```

Figure 22: Excerpt of a FirewallBuilder network object file created by Scylla.

This file can then be opened with *FirewallBuilder* and the user can select between classified flows. Every flow creates two rules by default, one for inbound and one for outbound traffic. For security reasons all flows are initially disabled. The network behind the firewall is grouped into an object called inside to make the ruleset easier to comprehend. Besides this Scylla generated *FirewallBuilder*

object library, there is also the standard library available which has many very useful building blocks for firewall policies that can simply be dragged and droped into the rule set. A tyipical Scylla generated rule set is depicted in figure 23.

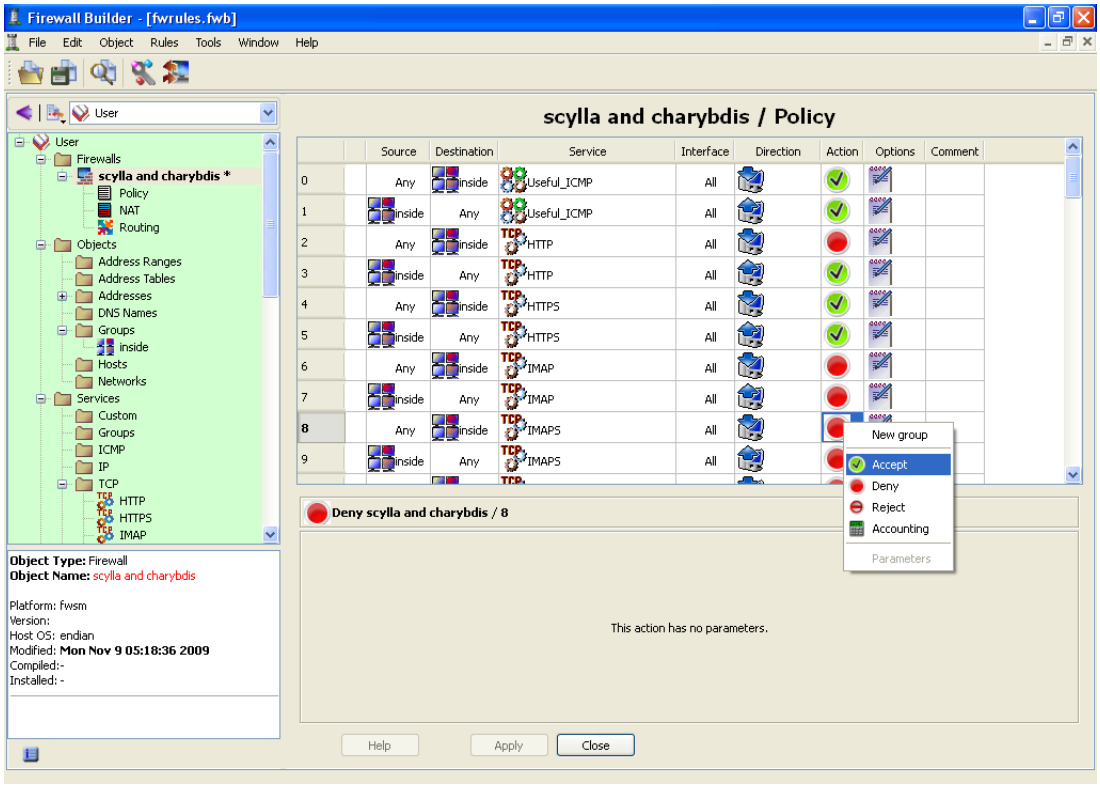

Figure 23: The exported rules opened in FirewallBuilder

When the user is satisfied with the created policies, the firewalling solution and operating system are selected as is shown in figure 24.

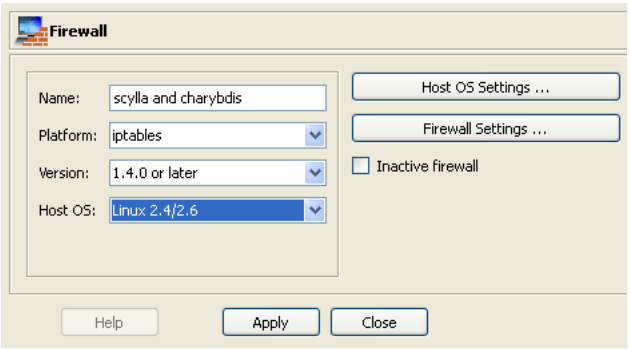

Figure 24: Selecting a target firewall solution.

A click on the compile policy icon starts the compilation process and creates the desired rule set in the same folder as the initial *FirewallBuilder* object file.

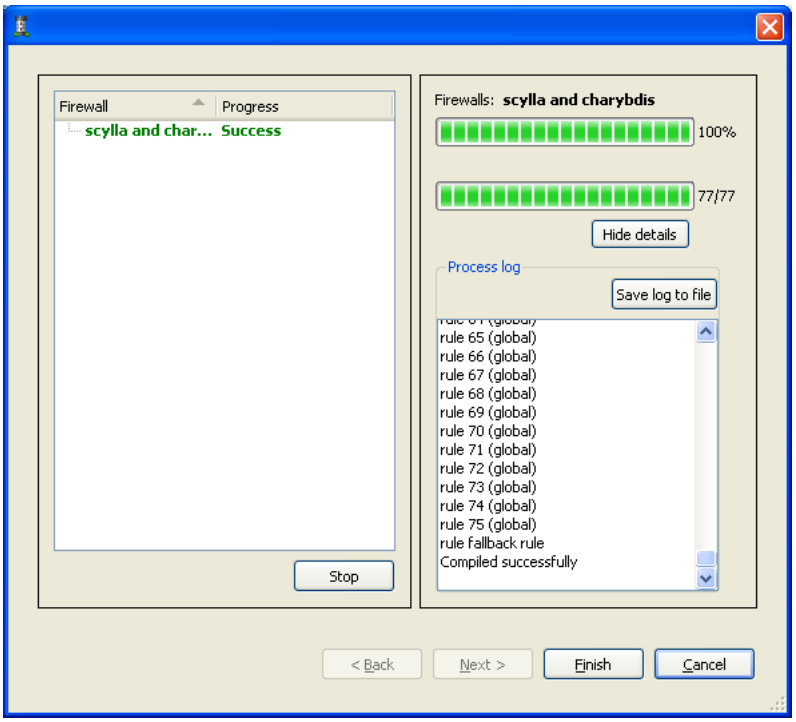

Figure 25: Compilation successful!

The user can now choose to either deploy this ruleset manually or use the functionality within *FirewallBuilder* to do so.

The rule set shown below in figure 26 is the final rule set for the school. The

user had to choose between 77 rules within FirewallBuilder from which a rule set with over 16800 lines was created.

rc.firewall.local

```
16778 "$IPFW" add 82030 set 1 drop    log udp  from 196.23.167.123  to any 2627 out  || exit 1
16779
16780 "$IPFW" add 82040 set 1 drop    log udp  from 196.23.167.124  to any 2627 out  || exit 1
16781
16782 "$IPFW" add 82050 set 1 drop    log udp  from 196.23.167.125  to any 2627 out  || exit 1
16783
16784 "$IPFW" add 82060 set 1 drop    log udp  from 196.23.167.127  to any 2627 out  || exit 1
16785
16786 "$IPFW" add 82070 set 1 drop    log udp  from 196.23.167.139  to any 2627 out  || exit 1
16787
16788 "$IPFW" add 82080 set 1 drop    log udp  from 196.23.167.221  to any 2627 out  || exit 1
16789
16790 #
16791 # Rule fallback rule
16792 #    fallback rule
16793 #
16794 "$IPFW" add 82090 set 1 drop   all  from any  to any      || exit 1
16795
16796
16797
16798 #
16799 # Epilog script
16800 #
16801
16802
16803 # End of epilog script
16804 #
16805 "$IPFW" set swap 0 1 || exit 1
16806 "$IPFW" delete set 1
16807
16808
```

Figure 26: Excerpt from an ipfw rule set created for FreeBSD

4.2 Results

Using passive traffic inspection as a datasource for automated firewall rule set generation was shown to be a positive approach, as is demonstrated by the proof of concept system implemented during the course of the project. The project is highly portable and successful compilation on most POSIX compliant but non-Microsoft operating systems can be achieved by simply uncommenting the #define _WINDOWS_ pragma in the traffic analyser's conf.h file.

The traffic analyser demonstrates unplanned and unanticipated functionalities beyond the initial scope of the project, that emerged from the scripting interface. One example of such an application is briefly elaborated upon in the following section 4.2.1.

Unfortunately the correctness of the rules generated by the rule generator cannot be proven, since the input on which their generation is based is limited by the comparatively short periods of time over which the system analyses traffic. It is this imperfect information that forms the basis for the rule generation, that does not allow to draw any deterministic conclusions on the correctness of the generated rules. The main problem is that there might potentially be an important service that was not observed during the analysis period, which would, as a consequence, not be considered in the rule set. Rule collision are however handled by *FirewallBuilder* and any rule set created by the system should therefore be free of them. Initial tests of the rule sets in the school's

network have been successful and no adverse effects have been experienced so far.

4.2.1 Detecting SYN scans

The traffic analyser is a powerful tool to investigate all aspects of the observed traffic. One possible application is the detection of SYN scans. Because a single SYN packet will not create a flow on its own, we have to resort to the FRAMES table in the traffic database to extract the relevant packets. An alternative way of isolating packets with certain criteria is to make use of the analyser's bpf interface and the corresponding -f or -F flags. An example shall be shown in the following.

SYN packets are isolated by issuing a query like:

```
create table synscan19623
as select distinct sIP1,sIP2,sIP3,sIP4,dIP1,dIP2,dIP3,dIP4
from frames where ipproto = 'TCP' and flags = 'SYN'
and sIP1 = 196 and sIP2 = 23 and dIP1 = 196 and dIP2 = 23
```

This will create a table showing all SYN packets exchanged on 196.23/16. This table can then in turn be visualised using the visualisation script, as can be seen in figure 27. Using the host / nslookup utility the host in the centre making connection attempts to all other machines on the subnet was identified as a dial-up line. The real extent of the scan can however only be seen when visualising the entire class A network segment, as can be seen in Figure 28. The dot graph took 47 minutes to complete and is 24 megabytes in size, while no indivudual nodes can be recognised on paper we can obviously zoom into the graph using a graphics package such as gimp. It is just shown here to illustrate how apparent such a scan is and how easily it can be detected. An interesting side effect of this approach to SYN scan detection is that even very slow running scans that could potentially evade some intrusion detection and prevention systems can be very clearly visualised as time over the observed period is basically flattened out.

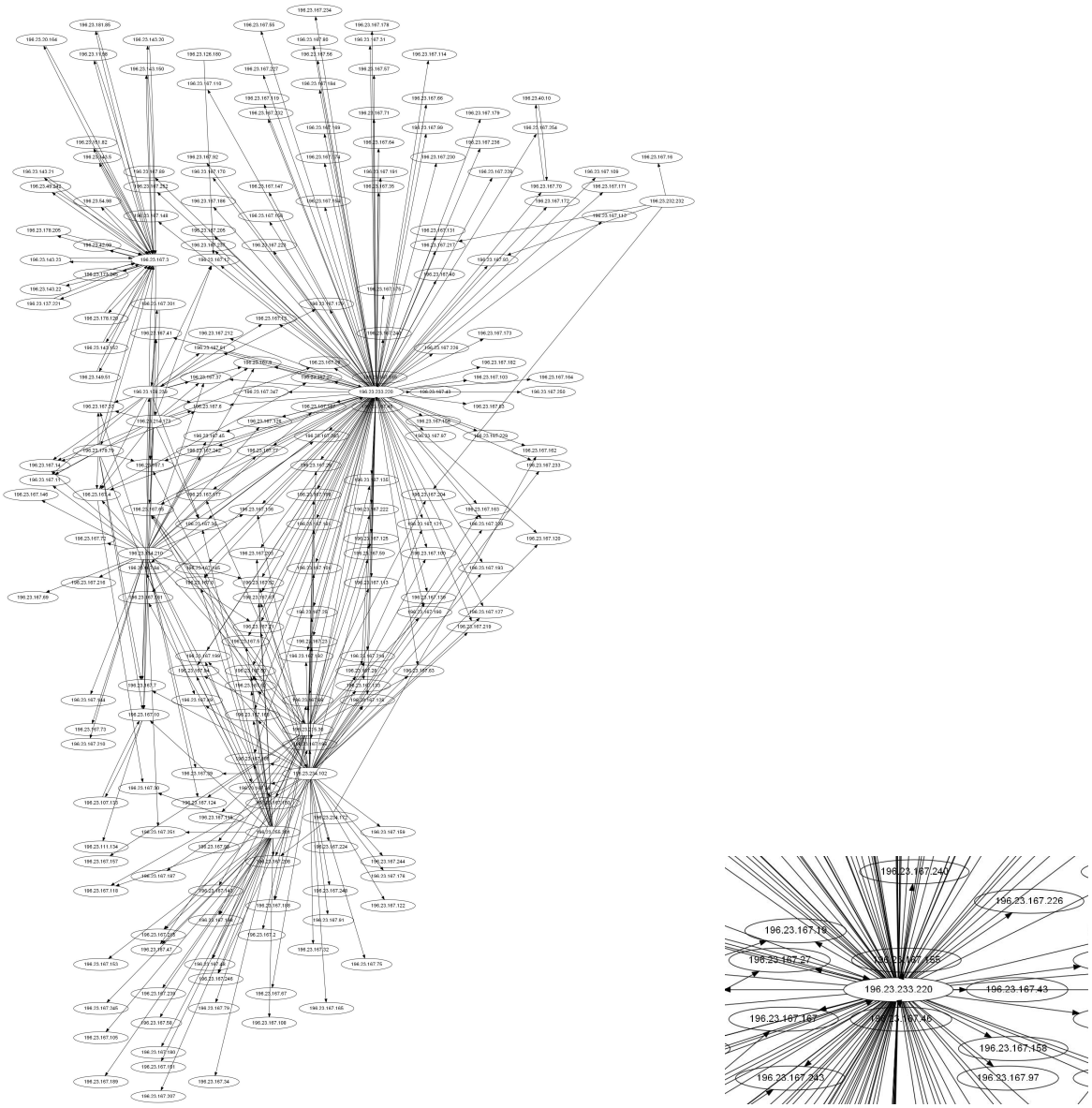

Figure 27: A SYN scan on 196.23/16

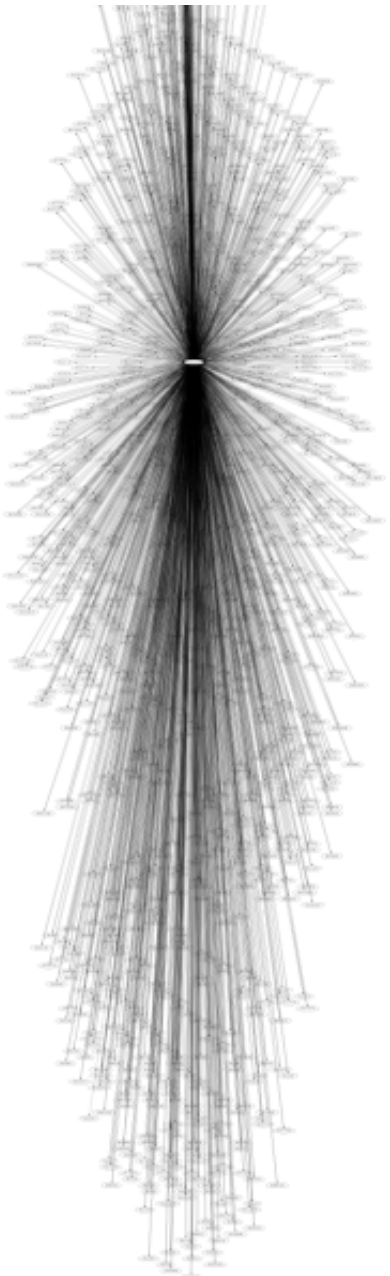

Figure 28: Part of the same SYN scan as in figure 27, but this time visualised on 196/8.

In summary this chapter provided a detailed overview of the performance testing of the system and found that with carefully chosen settings bottlenecks

within *SQLite* and operating system dependent terminal outputting routines could be overcome. Furthermore a full end-to-end case study presented the correctly functioning system. It was also shown how the system can be deployed for other non-firewall related tasks such as SYN scan detection and investigations into the topographical layout of a network or a specific service carried on the network.

5 Conclusion

It was shown that firewall rule set generation through passive traffic inspection is an overall positive approach, and that the goals of the project, as outlined in section 1.2, were met. The process of generating firewall rules still requires some human review and intervention, but this was expected from the very beginning of the project. Because the system matches all existing traffic by default it also creates rules for unwanted traffic that is present on the network, and it is these traffic flows that have to be pruned by a human user of the system. Employing a heuristic or some sort of fuzzy logic system to identify legitimate flows is simply not an option in a security context, as a misconfigured firewall will certainly only provide the illusion of network security.

The system was also shown to demonstrate some unanticipated functionalities, like SYN scan detection and network layout visualisation, that emerged from the pipeline created by the systems' components, that have proven useful beyond the initial scope of the project. Especially the scripting interface has proven to be extremely useful and will be more thoroughly explored in the future.

The system's to a large extent unattended operation was shown not only to be quicker and more convenient than manual firewall configuration, but is possibly more accurate and allows for faster turnaround in the deployment of new firewalling solutions. This also results in decreased risk and cost for organisations deploying such solutions.

5.1 Possible Future Extensions

One future extension that is definitely desirable is to extend the traffic analyser to include support for *NetFlow* and SNMP. This is mainly to accomodate for the aforementioned heterogentiy of networking equipment and by doing so broadening the range of targeted platforms and to increase versatility. *Netflow* as a fundamentally text-based format could be added elegantly via a lex / yacc grammar or if performance should turn out to be an issue, a small hand written LL parser.

The feasablity of creating optimized rule sets could be investigated, this would involve probing the *FirewallBuilder* policy compilers for existing functionality of this kind and modifying the rule generation process to take advantage of it. All the metrics that might be necessary for such an undertaking, like traffic volume or packet count could be extracted from the traffic database with relative ease.

A further very interesting extension would be to include an intrusion detection and prevention system, such as snort into the system and to consequntly configure it automatically as well. While this would certainly increase complexity, which is often non-beneficial to overall network security, in this particular case could proof advantageous since a firewall alone is just one network security component that by itself is not capable of preventing all forms of attack. One example that springs to mind is HTTP, often labeled as the universal firewall

traversal protocol, where an IDS will almost certainly increase overall network security.

Adding support for other database solutions would definitely benefit the system. Support for industrial strength databases like ORACLE could proof advantageous, especially when generating rule sets for WANs or VPNs. The subtle problem that the database connection during the traffic analysis would show up in the analysis file can easily be prevented by employing the relevant bpf filter. Whether or not networked database storage is feasible, largely depends on the load of the network in question, as the analysis process would almost certainly double the amount of traffic, since every packet creates a database entry. Furthermore alternative front-ends could be developed, web-based front ends based on php or perl would certainly go hand in hand with networked databases.

Another very exiting area of research in the department is packet filtering using a CUDA based GPGPU approach. Almost unbelievable speedups in traffic analysis have already been achieved. It can be expected that database solutions such as SQLite will be ported to these massively parallel platforms within only a few years. This would make traffic analysis and rule generation almost instantaneous and probably only unnoticeably longer than the period over which the traffic itself is observed.

References

[1] Firewall builder cookbook. Online: http://www.fwbuilder.org/guides/.

[2] Qt - a cross-platform application and ui framework. Online: http://www.qtsoftware.com.

[3] Sqlite. Online: http://www.sqlite.org.

[4] Tcpdump/libpcap public repository. Online: http://www.tcpdump.org.

[5] What is firewall builder. Online: http://fwbuilder.org/about.html.

[6] Winpcap: The windows packet capture library. Online: http://winpcap.org.

[7] Cisco ios ipsec accounting with cisco ios netflow. Technical report, Cisco Systems, 2004.

[8] Cisco cns netflow collection engine user guide, 5.0.3. Technical report, Cisco Systems, 2005.

[9] Introduction to cisco ios netflow - a technical overview. Technical report, Cisco Systems, 2007.

[10] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4*. Prentice Hall, 2006.

[11] Baek-Young Choi and Supratik Bhattacharyya. Observations on cisco sampled netflow. *SIGMETRICS Perform. Eval. Rev.*, 33:18 – 23, 2005.

[12] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better netflow. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 245–256, New York, NY, USA, 2004. ACM.

[13] Luis Martin Garcia. Programming with libpcap - sniffing the network from our own application. *hackin9*, 3:39, 2008.

[14] Terry Ogletree. *practical firewalls*. Que, 2000.

[15] Michael Owens. *The Definitive Guide to SQLite*. Apress, 2006.

[16] Karanjit S. Siyan and Tim Parker. *TCP Unleashed*. SAMS Publishing, 2002.

[17] Robin Sommer and Anja Feldmann. Netflow: information loss or win? In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 173–174, New York, NY, USA, 2002. ACM.

[18] Elizabeth D. Zwicky, Simon Cooper, and D. Brent Chapman. *Building Internet Firewalls*. O'Reilly, 2000.

Appendix A: Contents Of The CD

The enclosed CD containes this thesis itself and a mirror of the accompaning website. The sources of Scylla and Chraybdis are included within the mirrored site. For the latest versions of the sources please refer to http://www.cs.ru.ac.za/research/g05p3292/

```
+---thesis
|       coverpage.pdf
|       thesis.pdf
|
\---website
    |   blog.html
    |   default.css
    |   index.html
    |
    +---download
    |       about.pdf
    |       blunt-0.4.zip
    |       bluntPl-0.1.zip
    |       charybdis-0.5.7.zip
    |       charybdis1.wmv
    |       charybdis2.wmv
    |       litReview.pdf
    |       poster.pdf
    |       proposal.pdf
    |       scylla-0.4.3.zip
    |       scylla1.wmv
    |       scylla2.wmv
    |       scylla3.wmv
    |
    \---images
            bg02.jpg
            bg04.jpg
            bottom.png
            crimsonCrack.png
            menu.png
            top.png
```