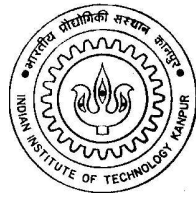


Supporting POP and IMAP in PickPacket

*A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology*

by

V V N Sudheer



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

June, 2005

Certificate

This is to certify that the work contained in the thesis entitled "*Supporting POP and IMAF in PickPacket*", by V V N Sudheer, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

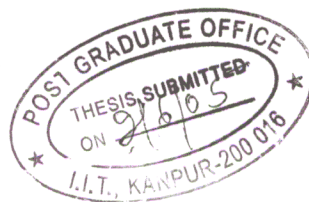
June, 2005

Anj Sanghi

(Dr. Dheeraj Sanghi)
Department of Computer Science &
Engineering,
Indian Institute of Technology,
Kanpur.

Deep

(Dr. Deepak Gupta)
Department of Computer Science &
Engineering,
Indian Institute of Technology,
Kanpur.



Sudheer

Abstract

Internet is a public medium for communication which can also be used for illegal activities. Therefore, there is a need to monitor network traffic. However, this monitoring should not compromise the privacy of individuals who are using the Internet for legal purposes. PickPacket - a network monitoring tool developed at IIT Kanpur can handle the conflicting issues of network monitoring and privacy through its judicious use. It is a passive tool in the sense that it neither injects any packet into the network nor delays any packet. PickPacket comprises of four components - *Configuration File Generator* helps the users in configuring the filtering parameters, *Filter* captures the packets from network, *Post-Processor* analyzes the captured data and *Data Viewer* for interactive rendering of the captured sessions.

PickPacket has support for HTTP, FTP, SMTP, Telnet, IRC and Yahoo-messenger protocols. It can filter traffic belonging to these protocols, reconstruct the sessions and display it to the user. POP and IMAP protocols are used to access the mails on a server. There is a need to monitor traffic of these mail access protocols. This thesis discusses an extension of PickPacket to IMAP and POP. The work involved in changing all components of the tool for the support of new protocols. Various tests were conducted to verify the correctness of the tool and to measure its performance.

Acknowledgments

I take this opportunity to express to place on record my gratitude towards my thesis supervisors Dr. Dheeraj Sanghi and Dr. Deepak Gupta for their invaluable guidance throughout my thesis work. It would have never been possible for me to take this project to completion without their innovative ideas and encouragement. The thesis is for a project that is financially supported by Ministry of Communications and Information Technology, Government of India. The support of the Ministry of Communications and Information Technology is duly acknowledged.

I also thank the other team members involved with the development of PickPacket for their cooperation especially Satya Srikanth helped me initially while understanding the Architecture of PickPacket. I also thank my project partner, Ananth, for his cooperation and innovative suggestions regarding the project. Ananth helped me alot while testing the filter. I will never forget those sleep less nights that we spent working and preparing things for the meetings. Devendar and Vinaya helped in changing the output structure of the post processor and developping a new web based GUI. I would like to thank Murthy for his valuable suggestions that he gave at the starting stages of my thesis.

I also wish to thank whole heartily all the faculty members of the Department of Computer Science and Engineering, IIT Kanpur for enhancing my knowledge. I would like to thank all my classmates for the moments I shared with them. Mtech2003 batch is one that I never forget in my life. I would also like to thank everyone in the Prabhu Goel Research Center for providing a nice and challenging work environment.

Finally, I would like to thank my parents and brother encouraging me all the times and taking me to this stage in life.

Contents

1	Introduction	1
1.1	Network Monitoring Tools	2
1.2	PickPacket	5
1.3	Organization of the Report	6
2	PickPacket: Architecture and Design	7
2.1	Architecture	7
2.2	Design	8
2.2.1	PickPacket Configuration File Generator	9
2.2.2	The PickPacket Filter	10
2.2.3	PickPacket Post-Processor	13
2.2.4	PickPacket Data Viewer	15
3	Design and Implementation of the POP Filter in PickPacket	17
3.1	POP Protocol Overview	17
3.1.1	Authentication Mechanisms in POP Protocol	18
3.1.2	Transactions in POP Protocol	20
3.2	POP Filter: Objective	21
3.3	POP Filter: Design and Implementation	22
4	Design and Implementation of the IMAP Filter in PickPacket	26
4.1	IMAP Protocol Overview	26
4.2	Differences between IMAP and POP	30
4.3	IMAP Filter: Objective	31

4.4	IMAP Filter: Design and Implementation	32
4.4.1	Parsing FETCH Response	33
4.5	IMAP Filter: Limitations	34
5	Post-Processor and Data Viewer for POP and IMAP	36
5.1	Post-Processor Design	36
5.2	Data Viewer Design	38
6	Correctness Verification and Performance Evaluation	43
6.1	Testing	43
6.2	Performance Evaluation	44
7	Conclusions and Future Work	47
7.1	Future Work	48
	References	49
A	Sample Configuration Files	51
A.1	Configuration File with Filtering Criteria (<i>.pcfg</i>)	51
A.2	Configuration File with Buffer Sizes(<i>.bcfg</i>)	60
B	A Sample POP Session	61

List of Tables

5.1	Format of meta-information file for POP and IMAP connections . . .	37
-----	--	----

List of Figures

1.1	Working of Hub and Switch	2
1.2	Working of Switch with SPAN port	3
2.1	Architecture of PickPacket	8
2.2	Filtering Levels	11
2.3	Basic Design of the PickPacket Filter	12
2.4	Post-Processing Design	14
3.1	POP State Diagram	18
3.2	Working of POP Filter	23
3.3	Connection match state diagram	25
4.1	IMAP State Diagram	27
4.2	Parser for Fetch Responce	33
5.1	Snapshot of Login Screen	38
5.2	Snapshot of Select Directory Screen	39
5.3	Snapshot of Select Connection Screen	39
5.4	Snapshot of Connection Filter Screen	40
5.5	Snapshot of Data Viewer for POP/IMAP connections	41
6.1	Test Setup for Performance Evaluation	45

Chapter 1

Introduction

The advent of the Internet has caused an ever increasing demand for computers in the last few years with enormous amount of information transferred everyday. The Internet has now become a major medium of communication all over the world. With increasing availability of the Internet to common man, it has become easy for criminals to exploit these resources and use them for illegal activities. This has resulted in the need for tools that can monitor the network traffic to detect and prevent such activities. Companies use these tools to safeguard their valuable data and to stop it from falling into wrong hands. However, the use of these tools should not compromise the privacy of individuals whose network communications are being monitored.

Monitoring tools are also useful in evaluating and diagnosing performance problems of servers and network components.

Many such tools are available in commercial as well as public domains. A brief discussion of network monitoring tools is presented in Section 1.1, along with a survey of some existing tools. PickPacket is a tool that monitors network traffic and stores those packets which match the user specified criteria. It provides a rich set of criteria for filtering the packets. It has support for many application protocols such as SMTP, HTTP, FTP, Telnet, IRC and Yahoo-Messenger. In this thesis, we describe the incorporation of support for email protocols (POP and IMAP) in PickPacket

1.1 Network Monitoring Tools

Network monitoring tools are also called sniffers. These tools are used to monitor data flowing across the network. They capture the network traffic based on some rules specified by the user. Network monitoring tools usually contain some protocol analysis capabilities that allow users to decode the captured data and analyse it. Monitoring tools can also be used to detect abnormal network activities or to find and fix problems in network.

The host machine on which the network monitoring tool is running must receive all network packets for it to filter. The network interface card, by default, would copy only those packets which are addressed to itself. If we want to read all packets, the interface card should operate in “promiscuous mode”. In this mode the interface card will copy all the packets that come up to the machine.

The data stream that we wish to monitor must be copied and made available to the sniffer host machine. In earlier LANs (Local Area Networks), where hubs were used, this was simple to achieve since hubs would copy every packet to all its ports. But, these days LANs use switches, which send the packet only on the port where the host is connected. The functioning of hub and switch is depicted in Figure 1.1. “Port mirroring” is a technique by which a port can be configured as a “span” port. Switches with these ports can copy all data flowing through any one port of that switch to this span port as shown in Figure 1.2. An alternate mechanism for this is using Ethernet taps, a special hardware that will serve the purpose without causing any interference in the data stream.

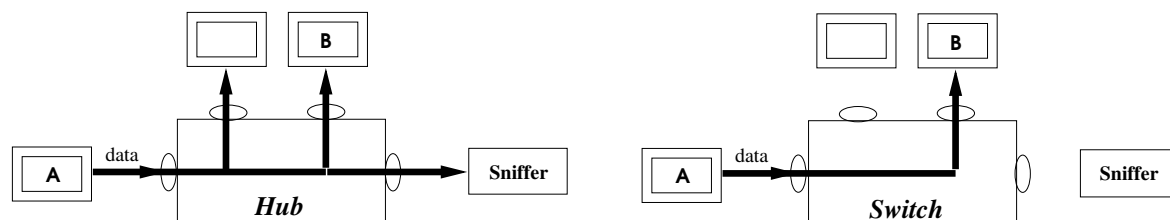


Figure 1.1: Working of Hub and Switch

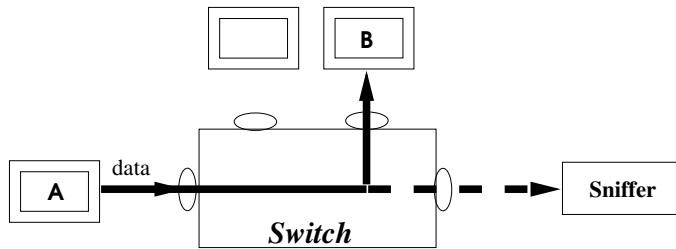


Figure 1.2: Working of Switch with SPAN port

Ideally speaking these sniffers should not be detectable, as they are passive listeners and do not inject any packet into the network. Sniffers should not affect the performance of the network, they should not slow down the network speed or they should not delay or drop any packet from the network.

Monitoring the network traffic should not compromise the privacy of individuals who are accessing data through the network. Filtering the captured data solves this problem. Filtering can be done ‘on-line’, while capturing, or ‘off-line’ after storing on the disk. Both of these are having their own advantages and disadvantages. For off-line filtering whole data flowing across the network has to be stored on the disk. If the monitoring is on a high speed link the disk will be filled up soon. On-line filtering helps in reducing storage required, but it needs machines with more compute power and memory for its operation.

Nowadays, sniffers often come with an embedded filter that can filter packets based on various criteria. The packets pass through different levels of filtering. The first level looks at the network parameters like IP addresses, protocols, and port numbers and drops the packets which do not match with the input parameters. This is generally supported by the kernel. BPF [14] (Berkeley Packet Filter) is an in-kernel packet filter that filters packets based on a directed acyclic Control Flow Graph method. BPF uses an interpreter for executing the filter code that assumes a pseudo machine with simple functionality akin to assembly language. The next filtering level searches for the application specific criteria like email-ids , host names etc. The final one looks at the application payload for a match in the content with the keywords. Packets which pass through all level of filtering will be stored on the

disk. These sniffers often come bundled with their own post-capture analysis and processing tools which extract meta information from the stored data and present it in a human readable form.

Packets are stored on the disk in a standard format. Libpcap is a standard packet capture library used to store packets on the disk. Many commercial and free post-processing and rendering tools are available that can analyze the packets stored by sniffers in the pcap format.

There are many sniffers which are available commercially and publicly. They come in various flavors with different capabilities, and for different operating systems. The following is a brief description of some of them. Detailed description of other tools is present in [18].

- *Ethereal* [4] can read packets either from the live network connection or from an already captured file. It has rich protocol analysis capabilities. However, it has limited filtering capabilities, it can filter packets based on IP addresses, port numbers. It comes in both read-only (protocol-analyzer) version and a capture (sniffing) version. The read-only version is used for analyzing data which has been captured already. This runs on UNIX as well as Windows.
- *Tcpdump* [10] is a UNIX based network protocol analyzer. UNIX based sniffers are generally built up on libpcap and/or BPF. *Tcpdump* is based on libpcap and BPF filters. The on-line filtering is limited, it can filter based on IP addresses and port numbers. *Windump* [3] is a version of *tcpdump* for windows that uses a libpcap-compatible library called WinCap.
- *Etherpeek* [5] is a commercial tool that can perform traffic monitoring as well as packet capture. *Etherpeek* can decode many application level protocols, but this decoding is off-line. The operation of this tool is limited to Ethernet networks. *Gigapeek* [6], an enhanced version of *Etherpeek* works at gigabit speeds. But, it needs a specialized hardware for its operation.
- *LANDecoder* [13] is another commercial protocol analyzer. it has good protocol analysis capabilities. The on-line filtering capabilities includes filtering based

on IP addresses and port numbers. User can also specify 32 bit patterns which should be present at specified offsets of selected packets.

- Microsoft's Win NT comes with an in-built program called "Network Monitor". This has been provided as a service called "Network Monitor Tools and Agent". It has the capabilities to capture frames coming from or going to the server and capture broadcast and multi-cast packets.
- *Carnivore* [7, 8, 20], a packet capturing system, has been developed by FBI. It is designed to collect information about the electronic communication to or from a specific user targeted in the investigation. Carnivore is quite different from other tools as it performs filtering of packet based on a wide range of filtering criteria specific to application level protocols. It functions through wire-taps across gateways and ISPs. It is capable of monitoring users who work on dynamic IP address based networks. It has capabilities to search the application-level content for specific strings.

These monitoring tools mainly focus on network management and trouble shooting aspects. Though they have good protocol analysis capabilities, they have limited on-line packet filtering capabilities. Carnivore has good filtering abilities but, it is not available to any one.

1.2 PickPacket

PickPacket is a Network Monitoring Tool that is being developed at Indian Institute of Technology Kanpur for the last three years. PickPacket can scan the network traffic and copy selected packets for further analysis. The selection criteria for the packets has been greatly improved compared to other existing tools of similar kind. The criterion can be specified at several layers of network protocol stack - IP addresses that belong to Network Layer, Port Numbers in Transport Layer, and application level parameters like email-ids, user names, URLs, and search strings.

PickPacket has support for application layer protocols like SMTP, Telnet, HTTP, FTP, RADIUS, IRC, Yahoo chat and instant messages. Users can specify criteria for

each application protocol separately. The basic frame work, design and implementation of application layer filters for Simple Mail Transfer Protocol (SMTP) and Telnet has been discussed in [12]. The design and implementation of application layer filter for Hyper Text Transfer Protocol (HTTP) and File Transfer Protocol (FTP) has been discussed in [18], for text string search in MIME-Encoded data in [1] and for application layer filter for the Remote Authentication Dial In User Service (RADIUS) Protocol in [11].

There are two levels of granularity, also called modes of operation, at which the PickPacket can capture the packets. The two modes of operation are called “PEN” and “FULL”. The Full mode of operation stores the whole connection, while in PEN mode a minimal amount of information about the connection is stored. Using these features judiciously will protect the privacy of users. The data stored on the disk is analyzed off-line and it is made available to the user with separate files for each connection. The tool will show summary of all connections as well as provide capability to view the details of each connection.

This thesis discusses the addition of support for the mail access protocols, POP and IMAP, in PickPacket. The mails accessed using these protocols can be captured based on user names, mail-ids and strings. The captured mails are reconstructed and the user is provided with capabilities to download or view the mails.

1.3 Organization of the Report

This report describes the extension of PickPacket to include monitoring of Post Office Protocol - Version3 (POP3) packets [17] and Internet Mail Access Protocol - Version4 (IMAP4) packets [2] in detail. Chapter 2 discusses the architecture and design of PickPacket. Chapter 3 describes the design and implementation of POP3 Filter. Chapter 4 discusses the design and implementation of IMAP Filter. Chapter 5 discusses about the post-processor and data viewer for handling POP and IMAP connections. Chapter 6 discusses the testing and performance results. The final chapter concludes the thesis with suggestions on future work.

Chapter 2

PickPacket: Architecture and Design

This chapter discusses the architecture and design of PickPacket. The design of each component is described briefly. Design and implementation issues are discussed in detail in References [1, 11, 12, 18].

2.1 Architecture

PickPacket can be viewed as an aggregate of four components ideally deployed on four different machines. These components are as follows.

- *PickPacket Configuration File Generator* is a JAVA based GUI for creating the configuration file. It can run either on a Windows or a Linux machine. The user can specify different criteria for filtering the data which is subsequently written to configuration files in a format that is understood by the filter.
- *PickPacket Filter* is deployed on a Linux machine. It takes the configuration file as input. It sniffs packets from the network and stores those packets which match the criteria specified in the configuration file. Filtering is done at different levels based on criteria like IP addresses, port numbers and application layer information.
- *PickPacket Post-Processor* runs on a Linux machine. It processes the packets stored on the disk and retrieves the meta-information from them and creates

a directory structure which is used by the Data Viewer.

- *PickPacket Data Viewer* is a web based GUI. It can be deployed on the same machine where the post-processor is running. It takes the directory created by the post-processor as input and displays the data in an interactive manner. The user can access the captured data through a web server.

An architectural view of PickPacket is shown in Figure 2.1 in which each of these components along with data-flow is shown.

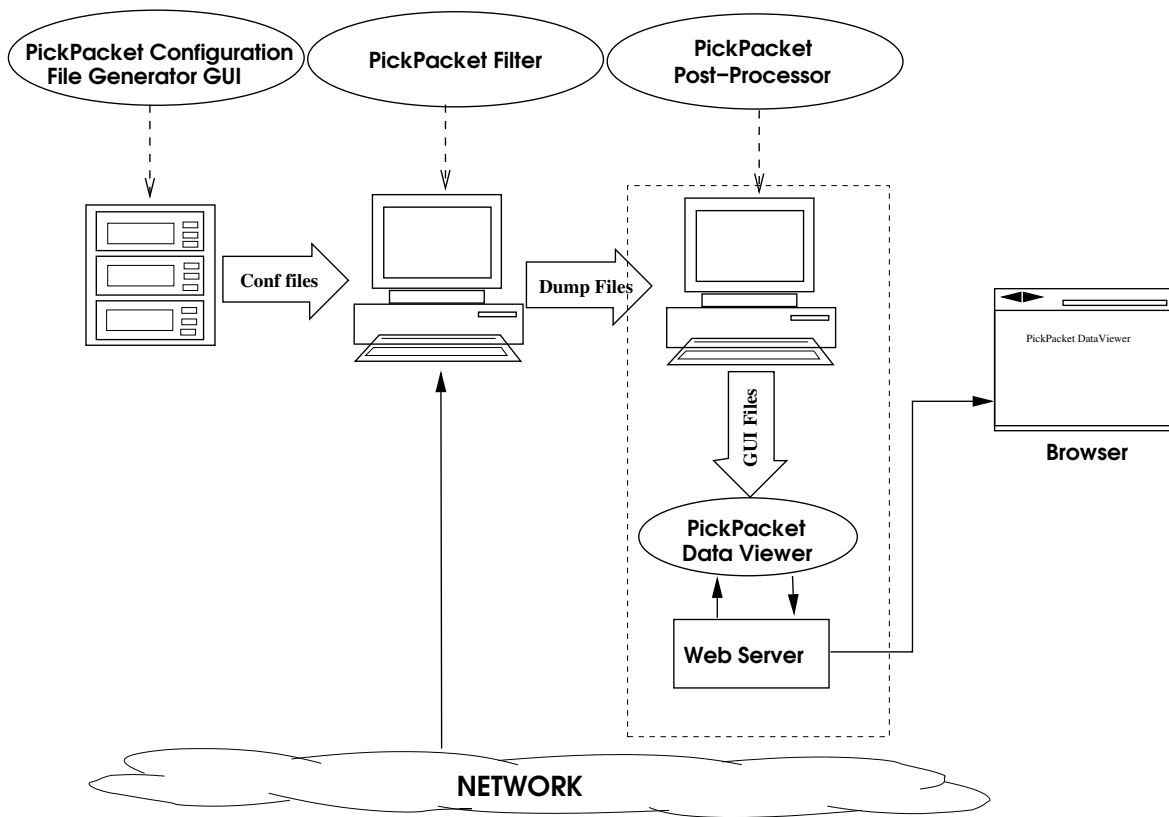


Figure 2.1: Architecture of PickPacket

2.2 Design

This section describes the design of each of the four components in PickPacket.

2.2.1 PickPacket Configuration File Generator

The PickPacket Configuration File Generator is a JAVA based graphical user interface where the user can specify the criteria by which the packets should be filtered. This will generate two files, one containing the filtering rules and the other containing the configuration parameters. Both of these files have the same base name with a different extension. The format of these files is similar to HTML, but with extra tags. Sample configuration files are given in *Appendix A*.

The first file with an extension *.pcfg* has three sections:

- The first section contains the settings of output files to store packets by PickPacket Filter. This section contains a File-Prefix which is used to generate the names of output files. File-Prefix is suffixed with a time stamp at which the file is generated. The names of output files will be generated automatically by the PickPacket Filter. The output file is changed periodically, so that the old output file can be transferred for further processing. The change of output file can be controlled either by specifying time or maximum size of file. This section can also contain the settings for multiple output file managers so that the captured packets can be stored in formats other than the default pcap [21] format.
- The source and destination IP addresses along with port numbers are there in the second section. The transport layer protocol and the application layer protocol for each of the different combinations are also indicated. These are used to demultiplex packets among different application layer filters.
- The third section comprises of multiple subsections, each containing the criteria corresponding to an application layer protocol. These criteria are used for filtering the packets at the application level filters. Here, the criteria for SMTP, HTTP, FTP, Telnet, IRC, YAHOO-chat and instant messenger protocols can be given. There is a subsection for specifying the strings which will be matched in the payload of all packets. The mode of operation of the filter, either PEN or FULL, is also mentioned for each application.

The second file with the extension *.bcfg* contains the number of simultaneous connections that can be monitored by the application filter and the maximum number of packets that can be stored before the criteria matches. These values are used for the allocation of buffers by the PickPacket Filter. The default value is set to 500 for each application protocol. This value should be chosen such that the system does not run out of memory and no connections are missed.

2.2.2 The PickPacket Filter

PickPacket Filter is the only on-line component. It sniffs packets from the network and writes the filtered packets onto disk. It filters packets based on the user specified criteria, packets that do not match the criteria will be ignored. Filter maintains the state for each connection that it encounters and stores packets if there is a match. When the data in a packet matches the criteria specified in the configuration file, all the previous packets of the connection (which are in memory) and all the future packets on this connection are going to be stored. Effectively, filter stores the connections which match the criteria instead of individual packets. This section briefly describes the design of PickPacket Filter.

Filtering of packets is done at various levels.

- Basic filtering on network parameters (IP addresses, port numbers).
- Application level filtering based on criteria like host names, user names, etc.
- Filtering based on content present in the application payload.

The use of in-kernel filters [14] made the basic filtering very efficient. Only those packets which match the network parameters are copied to the user space from the kernel space. Since the content of application can be best deciphered by the application itself, next two levels of filtering are combined.

Figure 2.2 illustrates various levels of filtering. Basic filtering is done on network parameters. Filter reads packets from the network and those packets which match the criteria will be sent to higher levels of filtering. Application layer filters look at

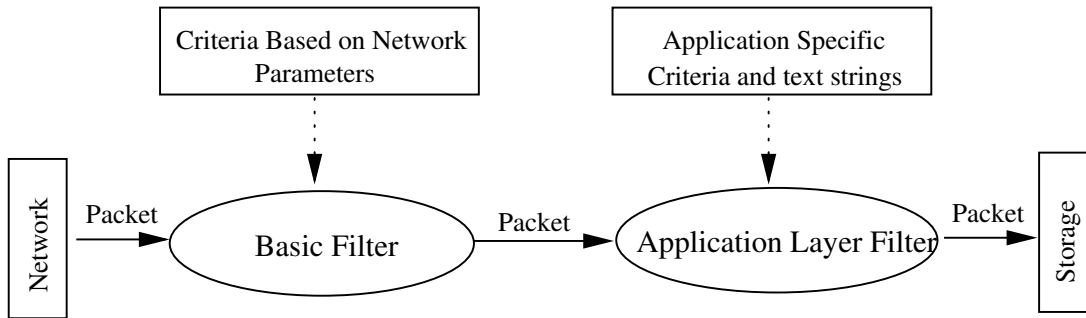


Figure 2.2: Filtering Levels

packets for application layer criteria as well as keywords. If there is a match they will be stored on to disk.

Each of the application layer protocol has a separate filter module. It looks for the corresponding application layer specific criteria. This design has an added advantage that it is easy to enhance the filter by adding filters for new application layer protocols. Demultiplexer module present between the basic filter and the application layer filters decides which application filter should receive the packet for further processing. The decision is made based on the rules present in the configuration file.

Application specific filtering includes application parameters and keywords. Application parameters are extracted from the packet and are checked for match. Keywords are matched in the application payload of packets. In case of communication over connection oriented protocol, this text search handles situations where the desired text is split across two or more packets.

Out of order packets are handled by *TCP Connection Manager* module present between demultiplexer and application filter. It maintains the sequence information of each connection and sends it to application filter. The design is efficient, and it will handle only those connections that are of interest to the application layer filter. Application layer filter can alert connection manager to maintain the sequence information for a connection.

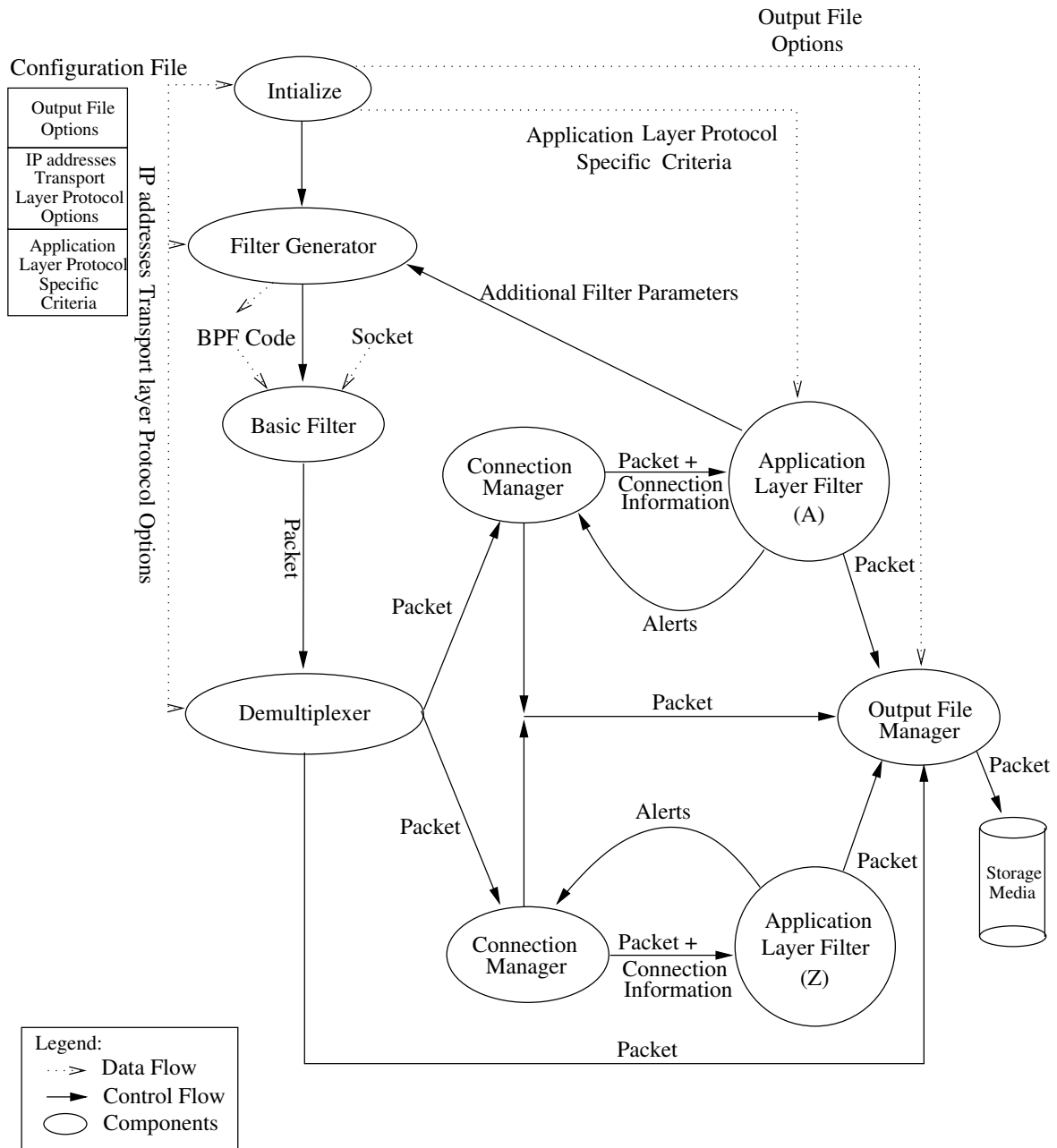


Figure 2.3: Basic Design of the PickPacket Filter

Figure 2.3 shows the major modules in the PickPacket Filter. The module *Initialize* is used for initialization of all other modules based on the configuration file. The *Filter Generator* module is used for generating the in-kernel BPF code. A facility to change the BPF code on-the-fly has been provided. Some of the application filters needs to change the BPF code during its operation. For example in FTP, during “PASSIVE” mode of file transfers, FTP filter [18] changes the BPF code. Application filters can call the functions that generate the filter code and change it. The *Output File Manager* module is responsible for storing the packets on to the disk. This module is responsible for changing the output file names periodically. The *Demultiplexer* is provided with a facility of calling the *Output File Manager* directly so that the filter directly store the packets without calling any application filter. The *Connection Manager* can also directly store packets to the disk. This is required if all the criteria are matched for a specific connection which is still open. More details of these modules can be found in [12].

The *output file manager* stores the output packets in *pcap* [21] file format. Utilities like *tcpdump* can also be used to view the captured data. This standard format allows us to use other tools for further analysis. A *pcap* file starts with 24 bytes of *pcap* file header and contains information related to *pcap* version and the network from which the packets were captured. This is followed by zero or more chunks of data. Every chunk has a packet header followed by the packet data. The packet header has three fields – the length and time of the packet when it was read from network and the length of the packet when it was saved to the disk.

PickPacket Filter provides a text string search library. This library uses the *Boyer-Moore* [19] string matching algorithm for searching strings. This library provides functions for both case sensitive and case insensitive search for text strings in packet data. This library is used by all the application filters for searching the text strings as well as application parameters.

2.2.3 PickPacket Post-Processor

PickPacket Post-Processor processes the output file which is generated by PickPacket Filter. It separates the packets in the output file into different connections

based on network layer and transport layer information. It also gathers the application layer specific meta information from these files. The objectives of Post-Processor are as follows:

- Packets in the output file which belong to connection should be separated based on communication tuple.
- The meta information about connections should be extracted and saved in a format understood by the user.

The Post-Processor has three modules: the *Connection Breaker*, the *Session Breaker* and the *Meta Information Gatherer*. These are shown in Figure 2.4.

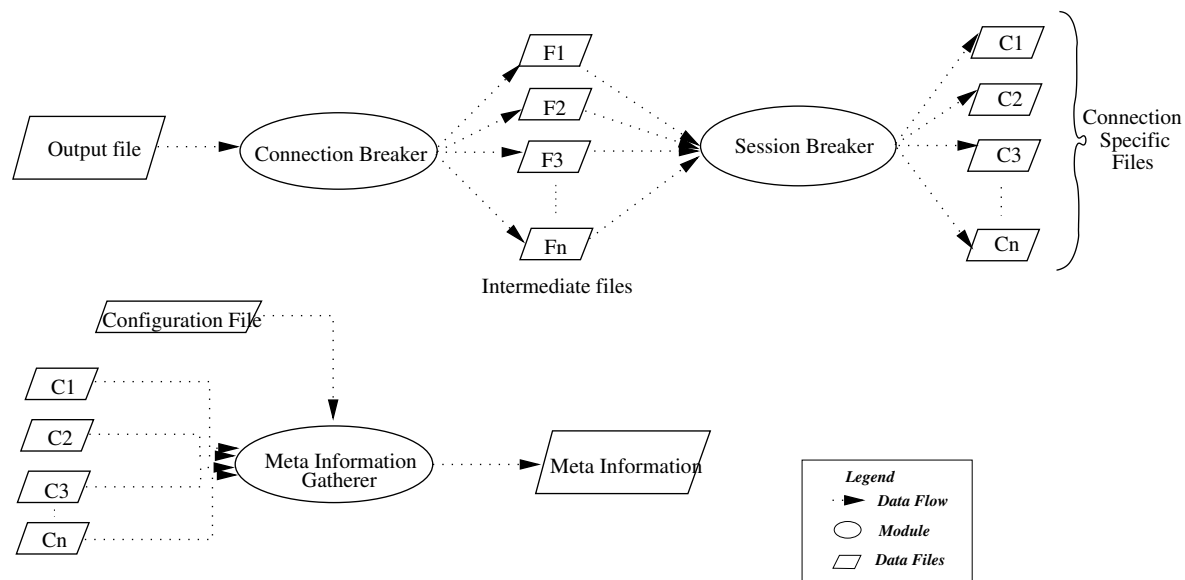


Figure 2.4: Post-Processing Design

The *Connection Breaker* module reads the output file of PickPacket Filter and separates it into different files based on the 4-tuple i.e. source and destination IP addresses and port numbers. The *Session Breaker* module takes each of these files and divides it further if more than one session exists with the same 4-tuple. This writes the packets in sorted order based on time stamp value, that is, the time at which the packets were read off the network. The *Meta Information Gathering*

Module takes configuration file and connection specific files as input. It retrieves the meta information for every connection. This module generates separate directory for each connection. The connection directory contains all the meta-information about that connection. The “tcpinfo” file contains the IP addresses, port numbers, transport protocol, application level protocol and the matched keywords in that connection. Depending on the application level protocol there is a file for storing the application specific meta data like hostnames and URI in HTTP connections, usernames and filenames for FTP connections etc. There are some more files created by this module which will be needed by PickPacket Data Viewer.

2.2.4 PickPacket Data Viewer

PickPacket Data Viewer is used for rendering the post-processed information. This is a web based graphical user interface written in PHP [9]. It is deployed along with a web server. Web server access the data through PHP scripts and serve the user requests. Data viewer can be deployed either on the same machine where the post-processing is done or on a different machine. This section gives a brief overview of data viewer.

Data viewer is provided with an authentication screen. The user can login in two modes “ADMIN mode” and “User mode”. Admin can add users, delete users and change their passwords. Users can only change their passwords. The Data Viewer has a configurable data directory parameter which contains all the high level directories generated by PickPacket Post-Processor. After logging in a user can select any of these high level directories. The Data Viewer reads the Meta information from that and lists all the connections showing MAC addresses, IP addresses, port numbers, Transport Protocol, Application Protocol, RADIUS User and keywords matched from the output of Post-Processor. Connection list can be sorted on any of these fields. User can change his configuration to show or remove some of of these fields. User can save the changes that are made in the configuration.

User can search among the captured connections. The search criteria includes network parameters, application parameters and keywords. User can search on all the fields that he specified in the configuration file. When a user sets a connection

filter for displaying the connection only those connections that match the criteria will be displayed. A User can get back all the connection by setting the connection filter to null i.e no filtering would be done.

On selecting a connection from the list of connections, the details of the connection are shown. The details include network parameters and application parameters. The Data Viewer provides user with facilities like downloading the e-mails, web pages accessed etc. The dialogue between communicating hosts can also be seen in a dialogue window. The configuration file used for the filtering can also be viewed. The connection-specific output file for each of the connection can also be downloaded separately.

Chapter 3

Design and Implementation of the POP Filter in PickPacket

This chapter discusses the design and implementation of the application layer filter for Post Office Protocol - Version3 (POP3) [17] in PickPacket. First, a brief overview of the protocol is given, with a focus on those features that are important for designing and implementing the filter. Then the design and implementation details of application layer protocol filter are presented.

3.1 POP Protocol Overview

The Post Office Protocol - Version3 is intended to access a mailbox of a user, located on a remote server. POP protocol allows a user to retrieve mails that the server is holding for the user. POP clients normally download the mails to local host and are deleted from the server. The clients can be configured not to delete the mails from the server.

Clients establish a TCP connection with the server on port 110. Server starts the session by sending a greeting message, followed by the client commands and the server responses. The POP session runs through a number of states during its lifetime. With the server greeting message, session enters the AUTHORIZATION state. In this state, the client must identify itself to the server. After a successful

authentication, the session enters the TRANSACTION state. In this state, the client requests actions on part of the POP server by sending various commands. POP server responds to each client command, this response can be either positive or negative. Eventually POP session enters the UPDATE state and session ends. These states and transitions between them are shown in Figure 3.1.

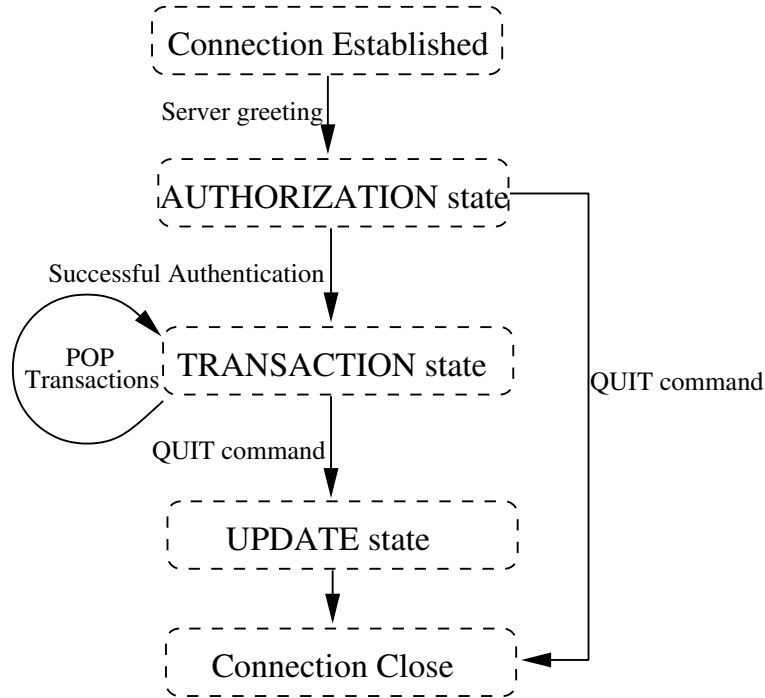


Figure 3.1: POP State Diagram

Section 3.1.1 discusses some of the authentication mechanisms used by POP. Transactions are the command-response pairs that occur in TRANSACTION state. POP transactions are briefly discussed in Section 3.1.2

3.1.1 Authentication Mechanisms in POP Protocol

POP client can authenticate to the server in a variety of ways. Following are some of the authentication mechanisms used by POP clients.

- The first mechanism is sending the username and password in plain text using “USER” and “PASS” commands.

USER <username>CRLF

PASS <password>CRLF

The client sends the USER command. If the server responds with a positive response, then the client issues a PASS command. Server uses username and password pair to check whether the client should be given access to the mailbox.

- *APOP* is a mechanism in which the client sends the username in plain text, and the MD5 digest of a time stamp and a shared secret between the client and the server.

APOP <username> <digest>CRLF

If the server supports APOP mechanism of authentication, it sends the time stamp in the initial greeting message. The “digest” parameter is a 16-octet value which is sent in hexadecimal format.

- *AUTH* [16] command specifies the authentication mechanism to be used with the server. The authentication protocol exchange consists of a series of server challenges and client responses that are specific to the authentication mechanism.

AUTH <mechanism>CRLF

Some of the supported authentication mechanisms are Kerberos-V4, CRAMMD5, LOGIN and GSSAPI.

After successful completion of authentication, the client can enter into TRANSACTION state. Before entering this state, the server opens and locks the mailbox belonging to this user.

3.1.2 Transactions in POP Protocol

POP transactions occur only in the TRANSACTION state. Retrieving mails, deleting mails from the mailbox, getting list of mails and asking for mailbox status, are various transactions of POP. The client sends a request command, the server responds with a positive or negative response. The general format of a client command is as follows:

```
<COMMAND> <zero or more arguments> CRLF
```

Some of the examples of client commands are “RETR”, “DELE” and “STAT”. “RETR” command will retrieve a mail from the server, “DELE” command marks a mail as deleted and “STAT” command requests the status of mailbox. The responses from the server are of the form:

```
+OK/-ERR <additional information> CRLF
```

Server responses can be multi-line; each of the line is terminated by a *CRLF* pair. The final line in the response consists of a termination octet (".", ASCII 46) and a CRLF pair.

On a successful “QUIT” command from TRANSACTION state the session enters into UPDATE state. In this state, the server does all its updations on the mailbox like deleting the marked mails and closes the mailbox followed by closing the connection. A QUIT command from AUTHORIZATION state directly closes the connection without entering into UPDATE state. A sample POP session is discussed in *Appendix B*.

■ Retrieve Command and Retrieve Reply

Retrieve command is used to fetch a mail from the server. This command can fetch one mail at a time. The format of retrieve command is as follows:

```
RETR <msg>CRLF
```

<msg> is message-number which the client is requesting. The server responds with a retrieve reply. The retrieve replies are of the form:

```
+OK <size> octetsCRLF
<mail data>
.
```

Positive responses for a ‘retrieve command’ are called retrieve replies. These responses are multi-line, first line contains +OK and the size of the mail data in octets. Here ‘octets’ string is mandatory by which we are finding the retrieve reply. The subsequent lines contain the mail body ending with a line containing a termination octet.

3.2 POP Filter: Objective

The POP Filter captures the POP packets flowing across the network according to the user specified criteria. Provisions have been made to specify usernames used for authentication, email-ids of users found in “To:” and “From:” addresses, and text strings to be searched in the mail body, including attachments. Filter will monitor POP connections based on these criteria.

POP uses a single connection to retrieve all the mails on the server. The main goal of the POP Filter is to capture only those mails which match at least one criteria. We first match the username. If username does not match, this connection is not to be monitored. If username matches (or username is not specified in criteria, meaning any username matches), then we look for addresses and strings. The mails which match the email addresses and strings specified in the configuration file are stored, rest are ignored.

The user can choose among the two modes of operation: “PEN” mode and “FULL” mode. In FULL mode, if a match occurs then the whole mail will be stored, but in PEN mode, only the mail envelopes will be stored. In PEN mode the body part of a packet containing both the header and the body will be replaced with ‘X’. There is limit on the number of packet to be stored before any criteria match occurs, which is specified by the user.

3.3 POP Filter: Design and Implementation

This section discusses the design and implementation of POP filter in PickPacket. POP Filter needs to monitor more than one connection simultaneously. All packets belonging to a single connection may not come in order. So, POP filter should maintain some state information corresponding to each connection that it comes across. This state information is initialized with the first packet of the connection that POP filter encounters. Whenever a packet arrives then the state information corresponding to that connection is updated.

In POP filter, username matching is done in AUTHORIZATION STATE of session. The usernames are retrieved from the authentication commands and are searched for a match with the once specified in the configuration file. If there is a match then the filter continues to monitor the connection, else it ignores the connection. Then in the TRANSACTION STATE, the transferred mails are captured based on mail-ids and strings. Only those mails which match the mail-ids and strings are stored.

Working of POP filter is shown in Figure 3.2. For every packet POP filter finds the type of command in the packet. If the command is an authentication command, then the username is retrieved and matched with the criteria. Match state is updated accordingly. After a username match, if a packet contains a 'retrieve reply' then filter starts monitoring a new mail. Filter checks for mail-ids and strings in this packet as well as subsequent data packets. Whenever there is a match the Mail_Flag is set and the whole mail is stored. Termination sequence in any packet ends the mail.

POP Filter maintains a structure for each connection. This structure contains the state information pertaining to that connection. Important members of this structure are `command_packet_list`, `command_dump_flag` and `conn_match_state`. The `command_packet_list` is a list of command packets stored before a match occurs. The `command_dump_flag` identifies whether the command packet needs to be stored in the memory or written to the disk. The `conn_match_state` indicates the matched state of the connection with respect to each criterion.

Whenever filter sees a new connection, it will allocate a structure for that connection and starts monitoring the connection. For every packet of that connection,

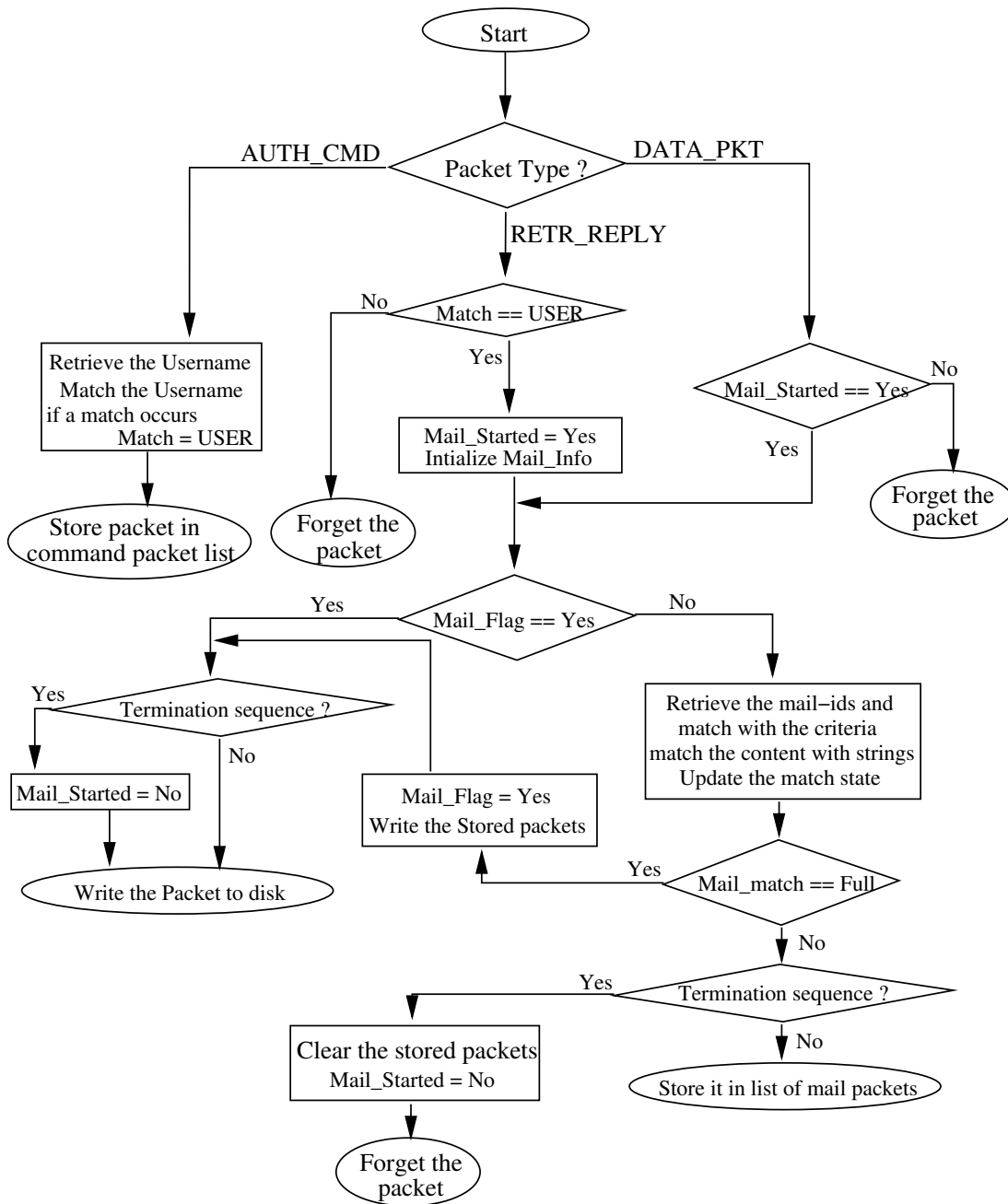


Figure 3.2: Working of POP Filter

it finds the packet type. Depending on the packet type and the connection match state it will take corresponding action.

Figure 3.3 shows how the match states are updated. There are two match states, one for the whole connection and the other for mail. For a new connection matching starts with the “Username”. If there is a match or the Username count is zero, the connection is marked as Username matched i.e. `conn_match_state` is set to `USERNAME_MATCHED`. Otherwise, the channel manager is alerted to ignore this connection. If the number of email-ids and number of strings turns out to be zero, the channel manager is alerted to store all packets of this connection without sending them to the filter. All command packets are stored in the command packet list in the structure maintained for the connection. When the first mail that matches the user specified criteria is stored, the command packets stored in the list are also written to the disk and a `command_dump_flag` is set. Once this flag is set, all the command packets are directly written to the disk and not stored in the list in main memory.

The next phase is selection of mails among the transferred mails. Matching in mail starts with a ‘retrieve reply’ and ends with a ‘termination octet’. Matched mails are written to the disk. First, POP filter extracts the “To:”, “From:” and “Cc:” fields from the mail envelope and searches for the user specified mail-ids. Mails which fail the mail-id matching are ignored. The string matching is done in the mail body. Encoded attachments are decoded and searched for the strings. If there is match then the whole mail is stored, else the packet is stored in the list of history packets for the mail. Meanwhile, if a termination octet appears in the mail body then the mail is ignored and the history packets are freed. This process continues until the connection is closed.

At any time the filter maintains information about a single mail for each POP connection. The filter has been implemented in such a way that this can be extended to maintain the information about more than one mail which would be able to handle out-of-order packets from the server.

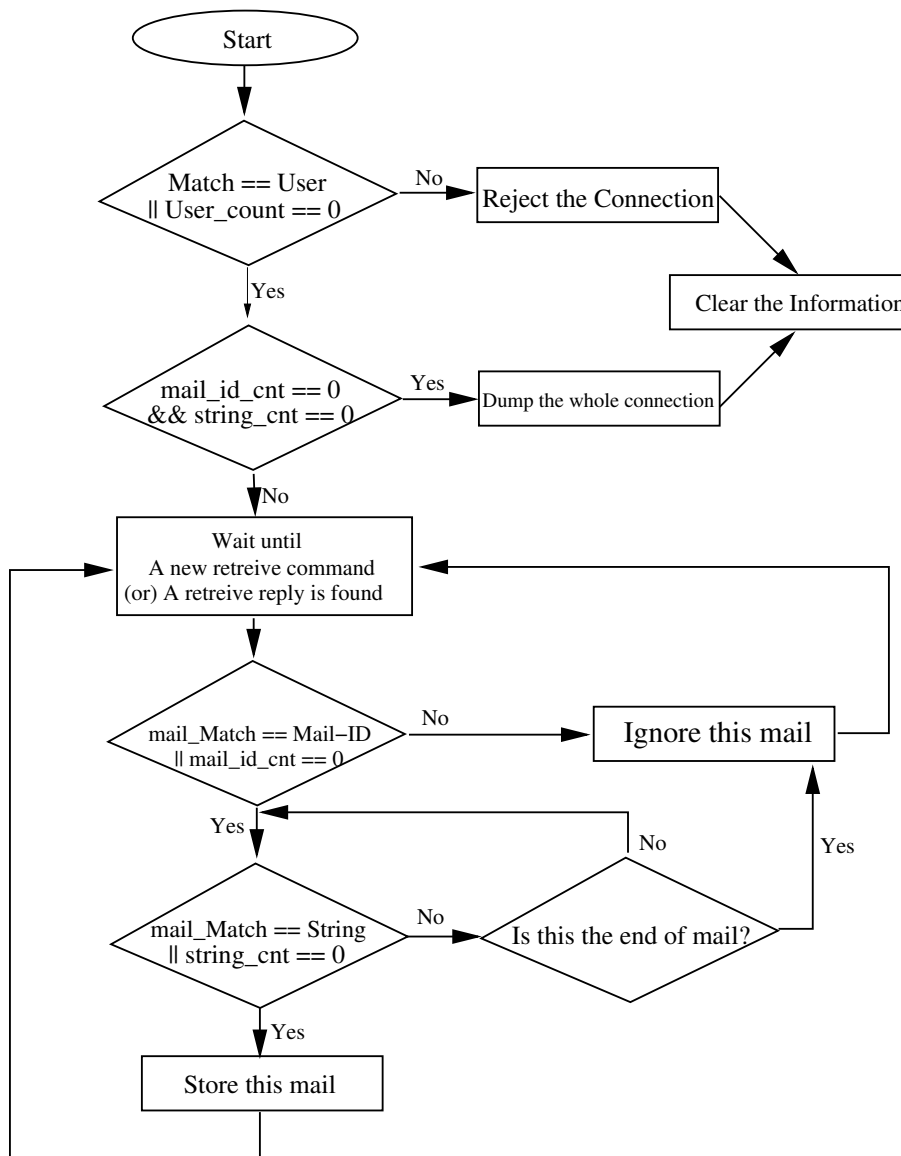


Figure 3.3: Connection match state diagram

Chapter 4

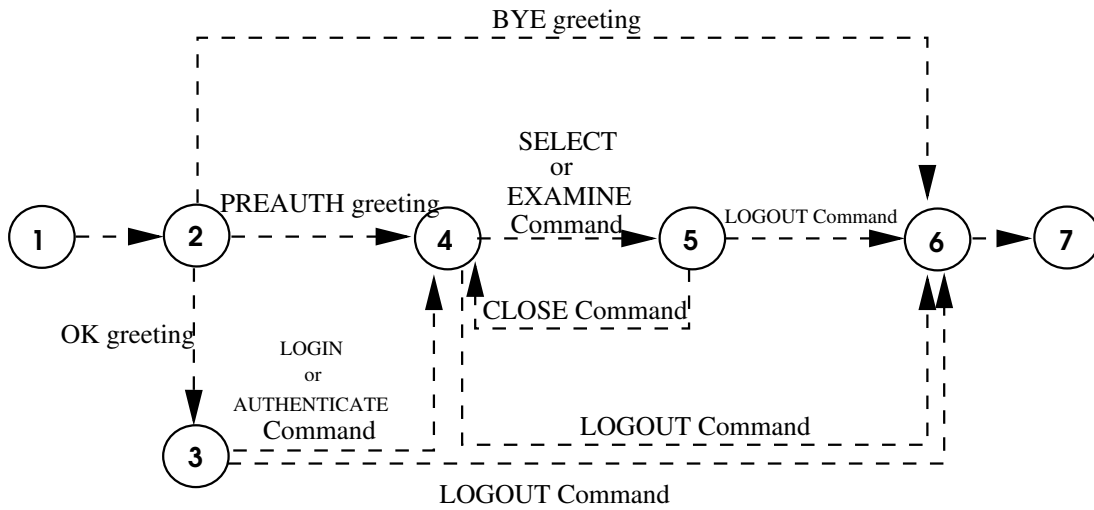
Design and Implementation of the IMAP Filter in PickPacket

This chapter discusses the design and implementation of the application layer filter in PickPacket for Internet Message Access Protocol - Version4 (IMAP) [2]. A brief introduction of IMAP protocol is given and the major differences between it and POP protocol are stated. This is followed by the design and implementation details of application layer protocol filter.

4.1 IMAP Protocol Overview

IMAP is a message access protocol used to access and manipulate electronic mail messages on the server. The Figure 4.1 shows different states and transitions between the states.

When a client establishes a TCP connection with IMAP server, the server sends a greeting message to the client. There are three types of server greeting messages: BYE, PREAUTH, and OK. BYE greeting tells the client that server is not accepting the client request and the connection is closed. Server sends a PREAUTH greeting if the client has already been authenticated by external means. Client need not do any authentication after a PREAUTH greeting from the server. After an OK greeting the client needs to authenticate itself to the server. Depending on the



- (1) TCP Connection Established State
- (2) Server greeting
- (3) Non Authenticated State
- (4) Authenticated State
- (5) Selected State
- (6) Logout State
- (7) Connection Close

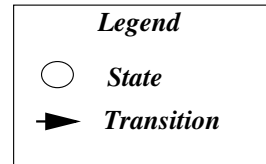


Figure 4.1: IMAP State Diagram

greeting message, the session enters Logout or Authenticated or Non-Authenticated states respectively.

Clients authenticate to the server by sending either a LOGIN or an AUTHENTICATE command. Clients enter Authenticated state from Non-Authenticated state on a successful authentication. SELECT and EXAMINE commands are used to select a mailbox on the server. On selecting a mailbox the state changes to Selected state. Clients can access and manipulate messages in the mailbox in the Selected state. CLOSE command closes an open mailbox. Client can select another mailbox after closing a mailbox. The CLOSE command changes the client state from Selected to Authenticated. Issuing a LOGOUT command from Non-Authenticated, Authenticated or Selected states leads the client to Logout state followed by closing of the connection.

All messages in a mailbox has a message sequence number. The first message has a sequence number '1', and every successive message is given the next number as the sequence number. Message sequence numbers are always consecutive, if a message is deleted in between then the message sequence numbers are updated accordingly. All messages in a mailbox have an 'unique identifier', it is a 64-bit number which is unique within that mailbox. Other attributes of messages include Message Flags, Internal Date, Size, Envelope and Body Structure. The 'Message Flags' contains flags like Seen, Deleted etc. Message flags are changed when ever a client accesses the message. These message flags can be changed explicitly by the client.

The client-server interactions in IMAP consist of a client command, server responses, and a server command completion response.

```
C: <tag> client command
S: * response1
S: * response2
...
S: <tag> command completion response
```

- All the client commands are prefixed with a tag. A tag is a small alphanumeric string, which is used by the server while sending command completion responses. Clients use different tags for different commands.

- The server response is prefixed with a '*'. The server responses are also known as untagged responses. These are used by the server to send either data or status to the client. The server response can be a command continuation request response. This response is used to request data from the client, this is prefixed with a '+'. On receiving a command continuation request response, client sends some data which does not contain any tags. These are generally used for authentication.
- The server command completion responses are also known as tagged responses. This response indicates to the client that the command is completed.

IMAP authentication is done using either a 'LOGIN' command or an 'AUTHENTICATE' command. The formats of these commands are:

```
<tag> LOGIN <username> <password>CRLF
```

```
<tag> AUTHENTICATE <mechanism>CRLF
```

LOGIN command sends the username and password in plain text. AUTHENTICATE command tells the server the mechanism to be used for authentication. There are many authentication mechanisms like KERBEROS_V4, LOGIN, CRAM_MD5 etc. These are discussed in [15]. After the AUTHENTICATE command there is an exchange of messages between client and server by which the authentication is completed.

IMAP has facility to access the messages in a variety of ways. Clients can request for message headers, message body, and attachments separately. Clients request the server for messages or parts of messages using a FETCH command. The format of which is as follows:

```
<tag> FETCH <message sequence numbers> ( list of data-items )
```

Fetch command can be used to request the server for a variety of data-items for which the server responds with those values. Some of the data-items that can be fetched are BODY[<section>], UID, BODYSTRUCTURE, ENVELOPE, FLAGS, RFC822, INTERNALDATE. BODYSTRUCTRE data-item gives a structure of body computed by the server by parsing the header fields. The data-item BODY[<section>]

is used to retrieve the mail body, the section tells which part of body to retrieve. This section can be empty, 'BODY[]', in which case the whole body along with attachments will be fetched. UID will request the server for the unique identifier of the message. Fetch responses are untagged data responses and contain pairs of data-item names and their values in parenthesis. Examples of fetch commands and responses are shown below:

```
C: a001 FETCH 3 (FLAGS RFC822.SIZE)
S: * 3 FETCH (FLAGS (\Seen,\Flagged) RFC822.SIZE 847)
S: a001 OK FETCH Completed
```

```
C: a002 FETCH 3 (UID BODY[HEADER])
S: * 3 FETCH (UID 12543789 BODY[HEADER] ( <mail header> ))
S: a002 OK FETCH Completed
```

IMAP has a command called 'UID'. UID is used with other commands like Fetch, Store and Copy. But, here unique identifier is used instead of message sequence numbers. An example of UID command with Fetch is:

```
<tag> UID FETCH 2345634 (FLAGS BODY[])
```

IMAP also supports some advanced features like using TLS for client-server communication. With a 'STARTTLS' command there is a TLS negotiation between the server and the client, all the communication followed by this command will be encrypted.

4.2 Differences between IMAP and POP

Both mail access protocols, POP and IMAP, have their own advantages and disadvantages. The following are some of the primary differences between the two protocols.

- The default behavior of POP clients is to delete accessed message from the server. In case of IMAP, the messages have to be deleted explicitly. POP clients have an option to tell the server to keep the message with it.

- POP server locks the mailbox while it is being accessed, no other client can access it until the lock is released. IMAP locking mechanism is improved so that mailbox can be accessed simultaneously and the integrity of mailbox is maintained.
- IMAP has facility for accessing only the message headers which reduces the time to notify the clients about new mail messages. POP clients can only request for the whole message.
- In IMAP, clients can also upload some messages to the server. The IMAP server's mailbox will be synchronized with the client's mailbox when ever the client connects or disconnects from the server where this feature is not available with POP. POP clients download all new messages to the local machine once it connects to the server.
- IMAP supports more than one mailbox where POP server can have only one mailbox "INBOX". POP clients maintains folders locally which are not known to server.

4.3 IMAP Filter: Objective

IMAP Filter captures packets of IMAP connections flowing across the network according to the user specified criteria. User can specify 'usernames' used for authentication, 'email-ids' of users found in "To:" and "From:" addresses, and 'text strings' as filtering criteria. IMAP filter captures connections in which the criteria matches.

IMAP uses a single connection to retrieve more than one message from the server. The main goal of the IMAP Filter is to capture only those messages which match the criteria. Once username is matched, then we are storing only those messages in which we find an email-id and a string specified by the user.

The user can choose between the two modes of operation: "PEN" mode and "FULL" mode. In FULL mode if a match occurs then whole message will be stored, but in PEN mode only the mail headers will be stored. In PEN mode, if a packet

contains both the header and message body, the body part will be replaced with 'X'.

User can specify a limit on the number of packets that are stored before a match occurs.

4.4 IMAP Filter: Design and Implementation

For each connection, filter first checks for the username. If the username matches with any one of the usernames specified by the user then the filter starts looking for messages, else ignores the connection. When ever a message is retrieved by the client, filter extracts the “To:” and “From:” addresses and checks for mail-id match. Mails with mail-id match are searched for the keywords in mail body and the decoded attachments. Fully matched emails are stored on the disk.

When ever a packet comes to the filter it finds the type of command in the packet. Depending on the command, corresponding action is taken and the state information is updated. If it is an authentication command, username is extracted from it and checked for any match. Fetch responses and the mail data packets are handled by a separate module which will check for email-id and string matching. These packets will be sent to Fetch Response Parser which is discussed in the Section 4.4.1. The sessions using TLS for communication cannot be checked for a match. STARTTLS command indicates to the filter that session is encrypted and the connection can be ignored.

The IMAP filter maintains a state information for each connection that it comes across. Each packet that belongs to this connection changes the state of the connection. This state information is initialized with the first packet in the connection. The important items in the state information are the state of the IMAP session, information about the mail that is being processed, and the matched state information of the connection. There is a global flag “Criteria_with_Zero_Usernames” which is set when there is at least one IMAP criteria without specifying username.

The username is matched when the connection is in Non-Authenticated state. The flag Criteria_with_Zero_Usernames is used to filter the connections in which

username is missing or we are unable to find username. For example, when the filter starts looking at a session from middle (we cannot get username) then if the flag is not set we can ignore the connection.

4.4.1 Parsing FETCH Response

Parsing fetch responses plays a key role while designing IMAP filter. The parser has been designed with an assumption that fetch responses containing message data for different messages come in different packets, i.e., fetch response with a message data does not come along with message data of another mail. But, a packet can contain more than one fetch response each containing no message data. In the subsequent discussion this parser is referred to as *FRParser*. Figure 4.2 shows the working of *FRParser*.

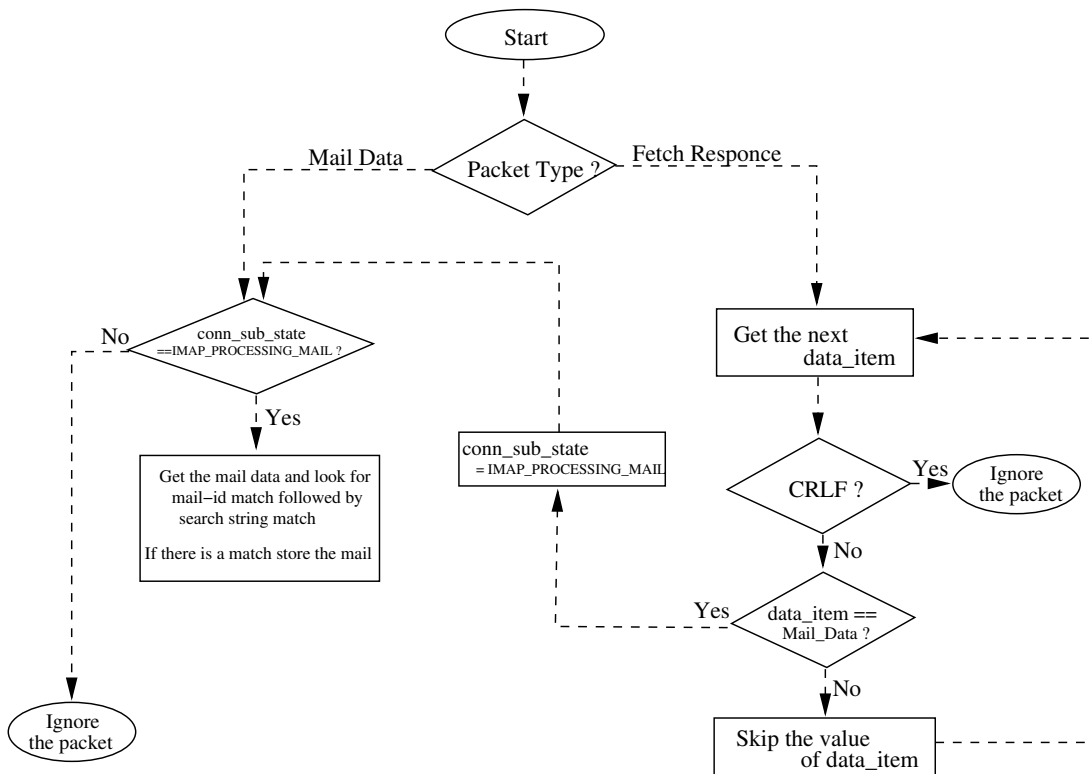


Figure 4.2: Parser for Fetch Response

There are two types of packets that come to the *FRParser*: the fetch responses and the mail data packets. Fetch responses are parsed to obtain the data-items and their values. If the data-item correspond to the mail data then the value contains the message. Whenever the data-item is mail data the `connection_sub_state` is set to “PROCESSING_MAIL_DATA”. The value part is mail content. Email-ids are matched in the “To:”, “From:” and “Cc:” fields of mail envelope. Keywords are matched in the mail body and decoded attachments. If the packet is a mail data packet and the `connection_sub_state` is PROCESSING_MAIL_DATA then email-ids and keywords are matched. The mails which match both email-ids and keywords are stored, the rest are ignored.

As IMAP client can request for mails in parts, we can combine them if they come in successive fetch commands. If response contains an attachment of a mail, the message sequence number and the unique identifier are compared with that of previous mail. If either of them are same, it means that they belong to the same mail. If the previous mail is stored then this attachment is also stored, otherwise it is ignored.

The data-items BODY[], BODY[TEXT/<section id>], RFC822, RFC822.TEXT come under the category of mail data. The *FRParser* iterates on the packet until it encounters a data-item that corresponds to the mail data. If the end of response is reached, then the packet is ignored.

4.5 IMAP Filter: Limitations

The following are the limitations of IMAP filter.

- Clients can request for different parts of mail at different times. Clients can request for the structure of mail, what attachments it has, what are the types of attachments and the attachments themselves. Consider a scenario in which a client requests for mail body and the MIME headers of attachments. Then another mail is accessed, followed by the attachments of the earlier mail. In this case, the attachments cannot be correlated with mail message, and hence ignored.

- Usernames cannot be found if the authentication mechanism involves some encryption. For example, username is encrypted in case of authentication mechanisms like KERBEROS and GSSAPI. However, the mechanisms in which encoded usernames are sent, they are decoded to find a match. LOGIN mechanism uses Base64 encoding, which the filter can decode before checking for usernames.

Chapter 5

Post-Processor and Data Viewer for POP and IMAP

This chapter describes the design for post-processor and data viewer of POP and IMAP packets. Post-processor retrieves meta-information for each connection, and the data viewer displays the captured data. This chapter has two sections, one for the post-processor and the other for data viewer.

5.1 Post-Processor Design

The Post-Processor reconstructs the whole connection and retrieves the meta information. In case of POP and IMAP connections, the meta data includes the “To:” addresses, “From:” addresses, Subject, matched keywords, and the mail content. Each of the captured mails will be written into a separate file in a format that is understood by mail clients. The meta information is written into a file named “popinfo” and “imapinfo” for POP and IMAP connections respectively. These files are in “.ini” file format which can be easily parsed by using standard libraries. The matched keywords for the connection, which is the union of the matched keywords per mail, are written into the “tcpipinfo” file. The format of both the files are similar as shown in Table 5.1:

The *Meta Information Gatherer* module of the Post-Processor demultiplexes the

```

username = "<captured username>"
[mail1]
to = "To addresses separated by spaces"
from = "From address"
cc = "Carbon copy addresses separated by spaces"
time = "time at which the message is captured"
mailfile = "filename.eml"
keywords = "matched keywords"
[mail2]
.
.
[mailn]

```

Table 5.1: Format of meta-information file for POP and IMAP connections

connections among different meta-handlers. There is one meta-handler for each supported protocol. POP meta-handler takes sorted connection-specific output file as its input. It reads each packet from the file and then finds the type of packet. If it is an authentication command, it will retrieve the username from that packet and will write it in “popinfo” file. Whenever it finds a ‘retrieve’ command or a ‘retrieve’ reply, it starts writing a new e-mail file. It will consider all successive packets as the mail data packets until it finds a "CR LF . CR LF" sequence (CR is Carriage Return and LF is Line Feed) in a packet or a new client command. The mail content will be decoded and checked for strings matched in the mail body as well as attachments. At the end of each mail, “To:”, “From:”, “Cc:” and Subject are retrieved from the e-mail file and written as a record in the “popinfo” file.


IMAP meta-handler behaves in a similar way, except that it will start a new mail on ‘fetch response containing mail data’. The successive mail data packets are considered as packets belonging to that mail. The mail is considered to have ended when it comes across a packet that does not have mail content. All the ‘fetch responses containing mail data’ are not considered as a new mail. If the mail content corresponds to an attachment, then it checks whether message sequence number or unique identifier is same as previous mail. If either of the two matches then this response is considered as a part of the previous mail and written in the

same email file. The username is found from the authentication commands: LOGIN and AUTHENTICATE. This will write similar records in “imapinfo” file, along with the username.

5.2 Data Viewer Design

In this section data viewer is explained in detail with snapshots. Explanation includes snapshots of each screen and usage of it. The italicized words in each screen are hyper links to other screens.

When an user connects to the data viewer, he has to authenticate himself by providing a username and password. Figure 5.1 is the snapshot of the login screen. After logging in, user needs to select a output file among the list of files in the data directory. As shown in Figure 5.2, output files in the data directory are listed.



PickPacket DataViewer (Web Edition)

Please login to start

Login:

Password:

Figure 5.1: Snapshot of Login Screen

After selecting a output file, data viewer provides the user with a screen containing a list of all connections in the output file. Data viewer displays IP addresses, port numbers, transport protocol, application protocol and the list of matched keywords for each connection. User can select among the connections for details. User can filter these connections by applying a *Connection Filter*. *Connection Filter* can filter the stored connections based on all the criteria parameters specified in the configuration file. We can restore all the connections by applying a null filter. The snapshot of select connection screen is shown in Figure 5.3 and *Connection Filter* screen in Figure 5.4. Once a connection is selected, the details of the connection are displayed.

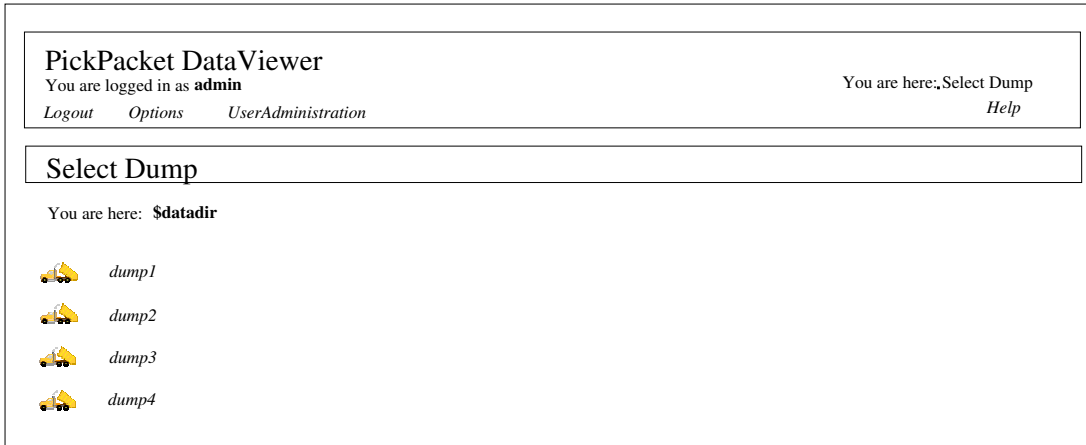


Figure 5.2: Snapshot of Select Directory Screen

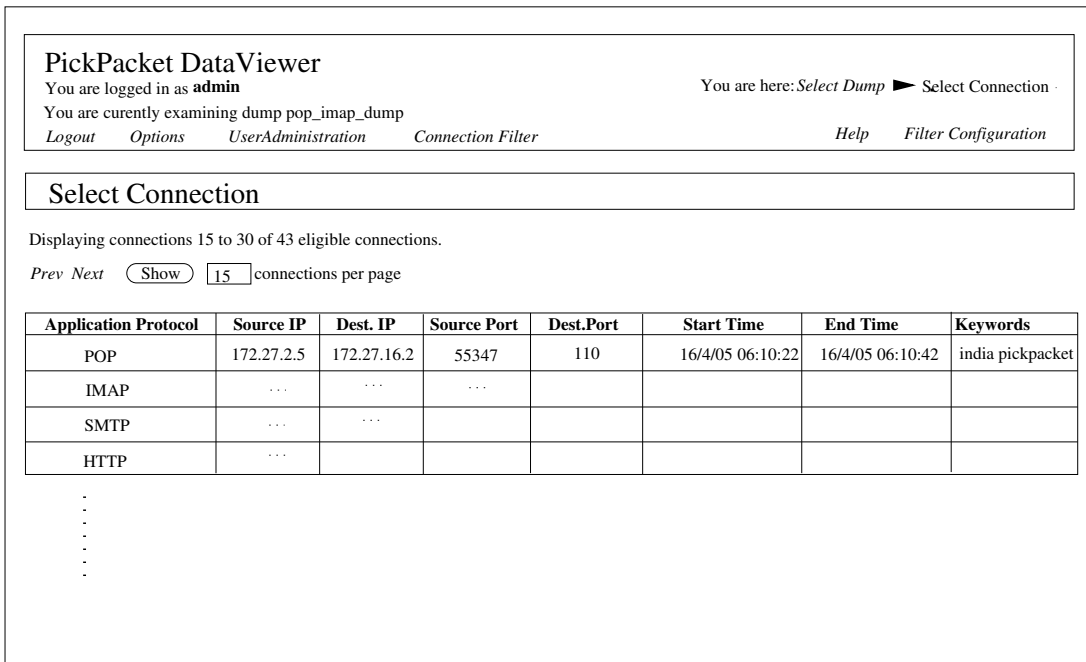


Figure 5.3: Snapshot of Select Connection Screen

PickPacket DataViewer
 You are logged in as **admin**
 You are currently examining dump pop_imap_dump
[Logout](#) [Options](#) [UserAdministration](#)

You are here: [Select Dump](#) ► [Select Connection](#) ► [Connection Filter](#)
[Help](#) [Filter Configuration](#)

Connection Filter

TCP/IP Level Filtering Options

Client IP addresses: Server IP addresses: RADIUS Users:
 Keywords:

Telnet connections? HTTP connections? SMTP connections?
 POP connections? IRC connections? FTP connections?
 YAHOO connections? IMAP connections?
 OTHER connections?

Telnet Filtering Options

Userids:

HTTP Filtering Options

Hostname: URIPath:

SMTP Filtering Options

Mail address:

POP Filtering Options

Username: Mail address:

IRC Filtering Options

Nickname: Channelname:

FTP Filtering Options

Username: Filename:

YAHOO Filtering Options

Username: Yahoo ID: Chatroom:

IMAP Filtering Options

Username: Mail address:

Figure 5.4: Snapshot of Connection Filter Screen

We have added support for POP and IMAP in data viewer. The details of POP/IMAP connection contains a list of mails with “To:” and “From:” addresses, Subject, and a link to the email file that can be opened or downloaded. The Figure 5.5 shows a snapshot of data viewer for a POP connection. IMAP detail screen is similar. The connection details screen consists of network parameters, meta information of application protocol and the keywords matched. Each row in the table corresponds to a mail transferred in that connection. From the message column of the table, the e-mail file can be downloaded.

PickPacket DataViewer
 You are logged in as **admin** You are here: [Select Dump](#) ► [Select Connection](#) ► [Connection Details](#)
 You are currently examining dump pop_imap_dump
[Logout](#) [Options](#) [UserAdministration](#)
[Help](#) [Filter Configuration](#)

Connection Details

Transport Protocol: TCP	Application Protocol: POP
Source MAC: 0:7:e9:9:a6:12	Dest. MAC: 0:e0:b0:64:71:17
RADIUS User:	Source IP: 172.27.2.5
Dest. IP: 172.27.16.2	Source Port: 50837
Dest. Port: 110	Start Time: 8/8/04 04:02:00
End Time: 8/8/04 04:02:06	Keywords: Milind

[View Connection Dialog](#) [Download Packet Dump](#)

POP specific connection details
Username: ananth

from	to	copy to	blind copy to	on subject	message	keywords
mail1@xyz.com	toaddress@xyz.com	ccaddress@xyz.com	bccaddress@xyz.com	This is subject	download	matched keywords
.....					
.....						
.....						

Figure 5.5: Snapshot of Data Viewer for POP/IMAP connections

The user can search amongst the captured connections for the specific keywords or email-ids or usernames. This feature increases the flexibility in using the data viewer. The user can download the connection-specific output file (file that contains the packets belonging to a single connection in *pcap* [21] format) by clicking on

Download Packet Dump, to use it with other publicly available tools. Data viewer can show the application protocol conversation between the client and the server. User can view it by clicking on *View Connection Dialog*. User can navigate amongst the screens by clicking on links in the top right corner.

Chapter 6

Correctness Verification and Performance Evaluation

In this chapter we discuss how we have tested the software after adding the new modules. In the performance evaluation we have whether the software is able to handle traffic at line speed on a 100 Mbps ethernet.

6.1 Testing

Correctness testing is done two phases. First one is to verify whether the new application filters are working properly. In the second phase, the whole software is tested for the correctness. Here, correctness means whether the packets stored by the *PickPacket Filter* are as per the criteria specified.

In the first phase, various tests were conducted to ensure the correctness of the newly added filters. POP and IMAP filters were tested specifying various criteria for filtering the connections. Post-Processor is tested to check whether the captured mails are reconstructed properly. Testing was done both for PEN mode and FULL mode of operation.

All the tests were conducted in the lab. The test setup includes two machines. One machine running PickPacket Filter. The other machine was used to generate the traffic by access various mail servers. Both these machines are connected to

a hub to the outside network. Filtering parameters of POP and IMAP filters are usernames, email-ids and keywords. Eight different types of criteria can be generated depending on whether each parameter is present or absent. Different configuration files are generated with all possible combinations of filtering parameters. We have run the filter with each of the configuration parameters on the same traffic. For each configuration, we know what connections are to be stored and what mails are to be stored in each connection. We found that the filter is storing the expected data in each test.

In the second phase, the tests conducted ensured that all components of software are working properly. Filter is tested specifying filtering parameters of all supported protocols. Traffic consisting of packets belonging to all supported protocols is generated and Filter is tested on this traffic. Traffic consists of few thousands of connections which we know apriori. Filtering criteria are specified such that a known number of different connections are stored among them. The behavior of filter on this traffic with the specified configuration is as expected, from this we conclude that PickPacket filter is working properly. Post-processor is tested to ensure that the reconstruction and meta data extraction are done correctly. Both the modes of operation are tested throughly.

6.2 Performance Evaluation

The experiments conducted for performance evaluation are similar to experiments described in [12, 18]. If the packets read by PickPacket filter with some filtering parameters and a sniffer which counts number of packets are same, then packets are not dropped because of application filter load. PickPacket filter was run on two machines simultaneously on the same network. First one simply counts the number of packets and the second one filters the packets based on user specified criteria.

Two instances of PickPacket filter were run on two identical machines. The configuration of both PCs was: Intel Pentium 3.6 G Hz CPU, 2GB RAM with PCI-X bus architecture. Both the machines were running on Redhat Linux 9 with 2.4.20-8 kernel and were connected to a 100 Mbps hub. PickPacket filter was run

on one machine without any application layer filtering criteria. It simply writes the packet to `/dev/null`. This is referred to as *counting sniffer*. PickPacket filter was run on the other machine using a configuration file containing filtering criteria. This is referred to as *filtering sniffer*. This instance was configured to write the packets on to disk. Test setup is as shown in Figure 6.1

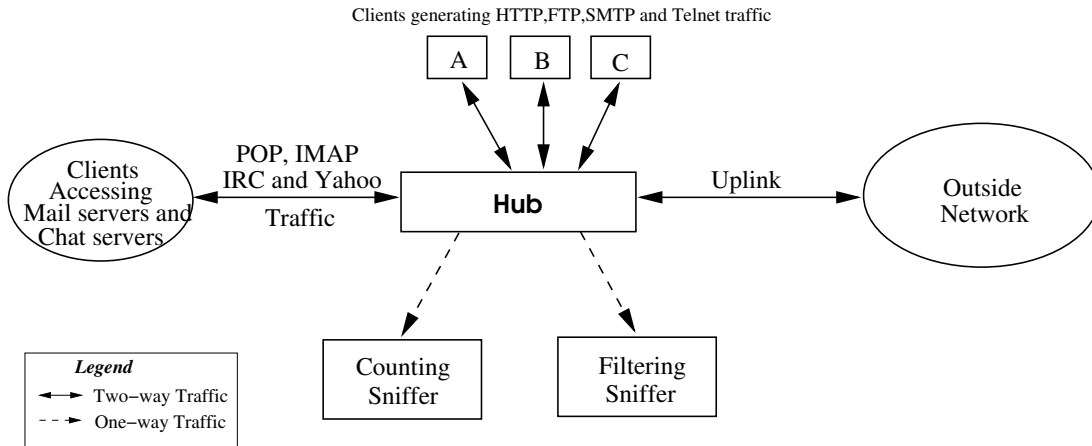


Figure 6.1: Test Setup for Performance Evaluation

Instead of evaluating the performance of PickPacket Filter only on new protocols, we have generated traffic that include packets of all supported protocols. We are intended in evaluating the performance of the software on a typical mix of flows that we find in Internet. Hence HTTP, FTP and SMTP form a large fraction of traffic generated. Traffic consisting of HTTP, FTP, SMTP and Telnet traffic was generated using scripts. POP and IMAP connections were opened from five different machines whose packets passed through the hub. This traffic also included some IRC and Yahoo Messenger packets.

Configuration files were generated containing the criteria specific for each supported protocol. The criteria for application protocols POP and IMAP included 100 usernames, 100 email-IDs and 100 keywords. The PickPacket Filter performance was tested with these configuration files. Both the sniffers (*counting sniffer* and *filtering sniffer*) are started manually at the same time and ran for the same time (6 minutes). In one experiment *Counting sniffer* processed 3533100 packets

and *filtering sniffer* processed 3527900 packets. The small difference in number of packets is due to the delay while starting the sniffers manually.

The testing is done generating traffic at slow rates and increasing the speed of traffic. We have found that PickPacket filter can handle traffic without loss of information at line speed on a 100 Mbps Ethernet segment. The line speed that can be achieved on a 100 Mbps network segment is around 65 Mbps, which we were able to handle. We have measured the usage of processor and memory while running the filter and found that they were less than 50% utilized even at peak. Therefore, the filter can easily run at much higher speeds.

Chapter 7

Conclusions and Future Work

PickPacket is a network monitoring tool that can capture packets flowing across the network and store some of the packets which match the user specified criteria. The criteria for filtering of packets ranges from network parameters like IP addresses and Port Numbers to application level parameters like Usernames, Email-Ids, URLs etc. The tool can operate in two modes capturing the whole connection or only application protocol meta information. Judicious use of PickPacket can help protect the privacy of individuals by capturing only the packets which match the user specified criteria. The storage format of packets is standard pcap format which is used by many publicly and commercially available tools. This adds the flexibility of using other processing and rendering tools, other than those provided by PickPacket.

We have added support for two mail access protocols “POP” and “IMAP” in all components of PickPacket. User can specify the user names, email-ids, and keywords as criteria based on which the PickPacket can capture emails. POP and IMAP application filters are designed and implemented. The Post-Processor was extended to extract meta-information and separates the mails accessed in a single POP/IMAP connection. Data Viewer which renders the captured data is updated with the support for new protocols. It provides the facilities to download the separated mails.

Various tests were conducted to test the functionality of POP Filter, IMAP Filter modules. We found that the PickPacket can handle traffic at line speed on a

100 Mbps Ethernet segment that is up to 70 Mbps. As CPU and memory are not utilized exhaustively, the filter can work at much higher speeds.

7.1 Future Work

PickPacket has support for application layer protocols like HTTP, FTP, SMTP, Telnet, IMAP, POP, IRC and Yahoo-messenger protocols. There is scope for extending the support for more protocols. PickPacket currently works on IPv4 packets. As use of IPv6 is growing, the support for this Internet Protocol needs to be added. Nowadays, HTTP servers and clients are using compression for the data transfer, i.e., servers are sending the requested data in a compressed format and clients will decompress it on the fly and display it to the user. Currently the HTTP module is unable to handle compressed HTTP data. As on-line compression becomes a norm for web servers, this functionality will become crucial in near future. Currently we support IRC and Yahoo chat protocols, MSN messenger is another popular chat application. Support for this needs to be added.

References

- [1] ADITYA, S. P. “Pickpacket: Design and Implementation of the HTTP postprocessor and MIME parser-decoder”, Dec 2002. BTP, Department of Computer Science and Engineering, IIT Kanpur, <http://www.cse.iitk.ac.in/research/btp2003/98316.html>.
- [2] CRISPIN, M. “Internet Message Transfer Protocol”. Tech. rep., 2003. <http://www.ietf.org/rfc/rfc3501.txt>.
- [3] DEGIOANNI, L., RISSO, F., AND VIANO, P. “Windump”. <http://netgroup-serv.polito.it/windump>.
- [4] ET AL., G. C. “Ethereal”. Available at <http://www.ethereal.com>.
- [5] “Etherpeek nx”. <http://www.wildpackets.com>.
- [6] “Gigapeek nx”. <http://www.wildpackets.com>.
- [7] GRAHAM, R. “carnivore faq”. <http://www.robertgraham.com/pubs/carnivore-faq.html>.
- [8] “How Carnivore Works”. <http://www.howstuffworks.com/carnivore.htm>.
- [9] “PHP:Hypertext Preprocessor”. <http://www.php.net>.
- [10] JACOBSON, V., LERES, C., AND MCCANNE, S. “tcpdump : A Network Monitoring and Packet Capturing Tool”. Available via anonymous FTP from <ftp://ftp.ee.lbl.gov> and www.tcpdump.org.

- [11] JAIN, S. K. “Implementation of RADIUS Support in Pickpacket”. Master’s thesis, Department of Computer Science and Engineering, IIT Kanpur, Apr 2003. <http://www.cse.iitk.ac.in/research/mtech2001/Y111122.html>.
- [12] KAPOOR, N. “Design and Implementation of a Network Monitoring Tool”. Master’s thesis, Department of Computer Science and Engineering, IIT Kanpur, Apr 2002. <http://www.cse.iitk.ac.in/research/mtech2000/Y011111.html>.
- [13] “LANDecoder”. <http://www.triticom.com>.
- [14] MCCANNE, S., AND JACOBSON, V. “The BSD Packet Filter: A New Architecture for User-level Packet Capture”. In *Proceedings of USENIX Winter Conference* (San Diego, California, Jan 1993), pp. 259–269.
- [15] MYERS, J. “IMAP4 Authentication Mechanisms”. Tech. rep., 1994. <http://www.ietf.org/rfc/rfc1731.txt>.
- [16] MYERS, J. “POP3 AUTHentication command”. Tech. rep., 1994. <http://www.ietf.org/rfc/rfc1734.txt>.
- [17] MYERS, J., AND ROSE, M. “Post Office Protocol”. Tech. rep., 2001. <http://www.ietf.org/rfc/rfc1939.txt>.
- [18] PANDE, B. “Design and Implementation of a Network Monitoring Tool”. Master’s thesis, Department of Computer Science and Engineering, IIT Kanpur, Sep 2002. <http://www.cse.iitk.ac.in/research/mtech2000/Y011104.html>.
- [19] R., B., AND MOORE, J. “A fast string searching algorithm”. In *Comm. ACM* 20 (1977), pp. 762–772.
- [20] SMITH, S. P., JR., H. P., KRENT, H., MENCİK, S., CRIDER, J. A., SHYONG, M., AND REYNOLDS, L. L. “Independent Technical Review of the Carnivore System”. Tech. rep., IIT Research Institute, Nov 2000. http://www.usdoj.gov/jmd/publications/carniv_entry.htm.
- [21] V., J., C., L., AND S., M. “*pcap - Packet Capture Library*”, 2001. Unix man page.

Appendix A

Sample Configuration Files

A.1 Configuration File with Filtering Criteria (*.pcfg*)

```
# This is a sample configuration file with filtering criteria
# A hash(#) is used for comments
# This file has several sections
# Sections start and end with tags similar to HTML.
# Tags within sections can start and end subsections or can be tag-value pairs.
# All the tags that are recognized appear in this file.
# First Section specifies the sizes and names of the dump files
# The Second Section specifies the source and destination IP ranges
# the source and destination ports, the protocol and the application
# that should handle these IPs and ports
# The next sections describe the application specific
# input criteria.
#*****First Section*****
<Output_File_Manager_Settings>
  <Default_Output_File_manager_Settings>
# File_Prefix is the name used to generate the dump filename suffixed with
# the time stamp at which the file is created
  File_Prefix=generaltest
```

```

# If the dump file has to be changed based on size then this field is having
#   value yes
#       Size_Based=yes
# This field exists when the Size_Based is yes this tell the size of dump
#       file in Mega Bytes
#       File_Size=100
# Time_Based attribute tells if the change of dump file is based on time also
#       Time_Based=yes
# This field exists when the Time_Based is yes this tell the time period in
#   minutes
#       Time_Period=60
#       </Default_Output_File_manager_Settings>
</Output_File_Manager_Settings>
*****End of First Section*****

```

```
C: RETR 1
```

```
S: +OK 120 octets
```

```
S: <the POP3 server sends message 1>
```

```
S: .
```

```
*****Second Section*****
```

```

# The basic criteria here are for the Device and
# SrcIP1:SrcIP2:DestIP1:DestIP2:SrcP1:SrcP2:DestP1:DestP2:ProtoA:App
# Should be read as For the range of source IP from SrcIP1 to SrcIP2
#       For associated ports from SrcP1 to SrcP2
# and For the range of destination IP from DestIP1 to DestIP2
#       For associated ports from DestP1 to DestP2
#       and FOR Protocol ProtoA
#       monitor connections according to Application App
# Protocols can be UDP or TCP
# Applications for TCP are
# SMTP, FTP, HTTP, TELNET, POP, IMAP, IRC, YAHOO, RADIUS, TEXT, DUMP_FULL, DUMP_PEN

```

```

# Applications for UDP are
#     DUMP_FULL, DUMP_PEN
# No further specs are required for DUMP kind of applications.
# Do not mix too many applications for clarity
# Take care that IPs Ports and applications do not conflict
<Basic_Criteria>
    DEVICE=eth0
Num_Of_Criteria=10
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:25-25:TCP:SMTP
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:20-20:TCP:FTP
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:21-21:TCP:FTP
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:110-110:TCP:POP
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:143-143:TCP:IMAP
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:23-23:TCP:TELNET
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:80-80:TCP:HTTP
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:143-143:TCP:TEXT
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1024-65535:TCP:DUMP_FULL
</Basic_Criteria>
#*****End of Second Section*****>

#*****Application Specific Specifications*****
# Here the criteria corresponding to different application level
# protocols are specified

#*****IMAP Specifications*****
<IMAP_Criteria>
    NUM_of_Criteria=2
    <Usernames>
        Num_of_Usernames=1
        Case-Sensitive=no
        Username=sudheerv

```

```
</Usernames>
<Search_Email_ID>
    Num_of_email_id=2
    Case-Sensitive=yes
    E-mail_ID=ananth
    E-mail_ID=deshaw
</Search_Email_ID>
<Search_Text_Strings>
    Num_of_Strings=0
</Search_Text_Strings>
<Usernames>
    Num_of_Usernames=1
    Case-Sensitive=no
    Username=sudheer
</Usernames>
<Search_Email_ID>
    Num_of_email_id=1
    Case-Sensitive=yes
    E-mail_ID=deepak
</Search_Email_ID>
<Search_Text_Strings>
    Num_of_Strings=2
    Case-Sensitive=no
    String=pickpacket
    String=IMAP
</Search_Text_Strings>
</IMAP_Criteria>
*****END of IMAP Specifications*****

*****POP Specifications*****
<POP_Criteria>
```

```
NUM_of_Criteria=2
<Usernames>
    Num_of_Usernames=1
    Case-Sensitive=no
    Username=ananth
</Usernames>
<Search_Email_ID>
    Num_of_email_id=2
    Case-Sensitive=yes
    E-mail_ID=sudheer
E-mail_ID=sybase
</Search_Email_ID>
<Search_Text_Strings>
    Num_of_Strings=0
</Search_Text_Strings>
<Usernames>
    Num_of_Usernames=1
    Case-Sensitive=no
    Username=jainbk
</Usernames>
<Search_Email_ID>
    Num_of_email_id=1
    Case-Sensitive=yes
    E-mail_ID=dheeraj
</Search_Email_ID>
<Search_Text_Strings>
    Num_of_Strings=2
    Case-Sensitive=no
    String=sachet
    String=POP
</Search_Text_Strings>
```

```
</POP_Criteria>
*****END of POP Specifications*****

*****SMTP Specifications*****
<SMTP_Configuration>
  <SMTP_Criteria>
    NUM_of_Criteria=2
    <Search_Email_ID>
      Num_of_email_id=1
      Case-Sensitive=yes
      E-mail_ID=sudheerv@cse.iitk.ac.in
    </Search_Email_ID>
    <Search_Text_Strings>
      Num_of_Strings=1
      Case-Sensitive=yes
      String=PickPacket
    </Search_Text_Strings>
    <Search_Email_ID>
      Num_of_email_id=2
      Case-Sensitive=yes
      E-mail_ID=ananth@iitk.ac.in
      E-mail_ID=jainbk@hotmail.com
    </Search_Email_ID>
    <Search_Text_Strings>
      Num_of_Strings=0
    </Search_Text_Strings>
  </SMTP_Criteria>
  Mode_Of_Operation=full
</SMTP_Configuration>
*****END of SMTP Specifications*****
```



```
*****FTP Specifications*****
<FTP_Configuration>
  <FTP_Criteria>
    NUM_of_Criteria=1
    <Usernames>
      Num_Of_Usernames=2
      Case-Sensitive=no
      Username=puneetk
      Username=jainbk
    </Usernames>
    <Filenames>
      Num_Of_Filenames=1
      Case-Sensitive=no
      Filename=test.txt
    </Filenames>
    <Search_Text_Strings>
      Num_Of_Strings=1
      Case-Sensitive=yes
      String=book secret
    </Search_Text_Strings>
  </FTP_Criteria>
  Monitor_FTP_Data=yes
  Mode_of_Operation=full
</FTP_Configuration>
*****END of FTP Specifications*****
```

```
*****HTTP Specifications*****
<HTTP_Configuration>
  <HTTP_Criteria>
    NUM_of_Criteria=1
  <Host>
```

```
        Num_Of_Hosts=1
        Case-Sensitive=no
        HOST=http://www.rediff.com
    </Host>
    <Path>
        Num_Of_Paths=1
        Case-Sensitive=yes
        PATH=cricket
    </Path>
    <Search_Text_Strings>
        Num_of_Strings=1
        Case-Sensitive=no
        String=neutral venu
    </Search_Text_Strings>
</HTTP_Criteria>
<Port_List>
    Num_of_Ports=1
    HTTP_Server_Port=80
</Port_List>
    Mode_Of_Operation=full
</HTTP_Configuration>
#*****END of HTTP Specifications*****
```

```
#*****TELNET Specifications*****
```

```
<TELNET_Configuration>
    <Usernames>
        Num_of_Usernames=1
        Case-Sensitive=yes
        Username=ankanand
    </Usernames>
    Mode_Of_Operation=full
```

```
</TELNET_Configuration>
*****END of TELNET Specifications*****
*****TEXT SEARCH Specifications*****
<TEXT_Configuration>
  <Search_Text_Strings>
    Num_of_Strings=1
    Case-Sensitive=no
    String=timesofindia
  </Search_Text_Strings>
  Mode_Of_Operation=pen
</TEXT_Configuration>
*****END of TEXT SEARCH Specifications*****
*****End Application Specific Specifications****
```

A.2 Configuration File with Buffer Sizes(*.bcfg*)

```
# The file contains the number of connections to open simultaneously
# for some applications
# and the number of packets to be stored per connection before a match
# occurs
<NUM_CONNECTIONS>
    NUM_CONNECTIONS=10
    NUM_SMTP_CONNECTIONS=500
    NUM_FTP_CONNECTIONS=500
    NUM_HTTP_CONNECTIONS=500
    NUM_TELNET_CONNECTIONS=500
    NUM_TEXT_CONNECTIONS=500
    NUM_RADIUS_CONNECTIONS=500
    NUM_POP_CONNECTIONS=500
    NUM_IMAP_CONNECTIONS=500
    NUM_IRC_CONNECTIONS=500
    NUM_YAHOO_CONNECTIONS=500
</NUM_CONNECTIONS>
Num_of_IMAP_Stored_Packets=100
Num_of_POP_Stored_Packets=100
Num_of_SMTP_Stored_Packets=100
Num_of_FTP_Stored_Packets=100
Num_of_HTTP_Stored_Packets=100
Num_of_TEXT_Stored_Packets=100
```

Appendix B

A Sample POP Session

```
# This is a sample POP session which gives a brief description of POP protocol.
# If the line starts with a '#' means some explanation.
# S: means Server reply (or) Server status
# C: means a command send by client (or) Client status

S: <waiting on port 110 for connection>
C: <open a new connection with server>
S: +OK POP3 mailhost v2001.78rh server ready

# The server listens on TCP port 110 and when the connection is open it sends a
# greeting message that the server is ready
# Now the client is in AUTHORIZATION State

C: AUTH LOGIN
S: + VXNlciBOYW11AA== # ''User Name'' in base64
C: c3VkaGV1cnY=
S: + UGFzc3dvcmQA # ''Password'' in base64
C: cGFzc3dvcmQ=
S: +OK Mailbox open, 192 messages
```

```
# This is an AUTH command with LOGIN as mechanism, in this mechanism the server
# requests for Username first followed by password and the client send both.
# The whole conversation happens base64 encoded
# The Client enters the TRANSACTION State
```

```
C: STAT
S: +OK 192 32577696
```

```
# The client asks for the status of the mailbox and the server reply with +OK
# and number of messages and mailbox size
```

```
C: LIST
S: +OK Mailbox scan listing follows
S: 1 7818
S: 2 338656
...
S: 192 325846
S: .
```

```
# Followed by the listing of messages, observe that there is a single '.'
# in a line indicates the end of server response.
```

```
C: RETR 1 # RETR command
S: +OK 7818 octets # RETR reply format is +OK <message size> octets
S: <message 1>
S: .
```

```
# This RETR command is to request the server for a message.
# This client request and the corresponding server
# response form a single transaction.
```

C: DELE 1
S: +OK message 1 deleted

This will mark the message 1 as deleted.

C: RETR 2
S: +OK 338656 octets
S: <message 2>
S: .
C: QUIT
S: +OK Sayonara

now the server enters the UPDATE state and deletes the
messages marked as deleted

C: <close connection>
S: <wait for new connection>