PickPacket: Resolving IPv4 Design Issues to make Compatible with IPv6 through the Support of Chat Protocols

Sudan Jha
PhD Scholar
Associate Professor, Principal
Eastern College of Engineering
Biratnagar, Nepal

Abstract — Internet media is quite popular for the electronic transfer of both business and personal information. But, the same media can be and has been used for unlawful activities. This demands the need for highly customizable network monitoring tools to capture suspected communications over the network and to analyze them. However, electronic surveillance may violate the rights of privacy, free speech and association, PickPacket - a network monitoring tool, can handle the conflicting is¬sues of network monitoring and privacy through its judicious use, PickPacket has four components - Configuration File Generator for assisting the users in setting up the filtering parameters, Filter for capturing the packets in the network, Postprocessor for analyzing the output files and Data Viewer for interactive displaying of the captured sessions.

Earlier version of PickPacket had support for four application protocols - SMTP, HTTP, FTP and Telnet, Chat protocols, by which a group of users form a network to communicate information among them, have gained popularity in the last few years. Active use of these protocols on the Internet motivated the need for support of chatting protocols in PickPacket, This paper work discusses extension of PickPacket for two chatting protocols (IEC and Yahoo Messenger), all components of the PickPacket have been upgraded for the support of new protocols, PickPacket has been tested for correctness and performance measurement.

Keywords—Internet Media, Networks; SMTP, HTTP, FTP; TELNET protocols, EAP protocols, IEEE 802.1x protocol with EAP-TTLS; Mutual Authentication;

I. INTRODUCTION

No doubt, Internet has become a key resource of information. On the other hand, it is also accepted that Internet has been an ease of medium by terrorists, criminals and others to communicate information about unlawful activities. Various Companies' intellectual property is not safe and they are indulged with their competitors. It has become necessary to rethink about network monitoring tools very sensitively to capture suspected communications over the network and to analyze them. There is a compulsion of need for development of sophisticated tools that can monitor and detect undesirable communications over the network.

Monitoring tools perform their task by sniffing packets from the network and filtering them on the basis of user specified rules. The tools like *Packet Filters* are used to fix offset packet information like IP addresses and port numbers for filtering. On the other hand, *Network monitoring tools* simply filter the packets based on the complex rules and perform post-capture analysis of collected traffic. They realize applications, and then search through packet application data. The following section describes working of network monitoring tools in general and also mentions various commercial and non-commercial tools available publicly. However, electronic surveillance conflicts with the rights of privacy, free speech and association. Section 1.2 explains how *PickPacket*, a network monitoring tool, works and how it addresses the conflicting requirements of privacy preservation and intelligence gathering, Section 1.3 gives the motivation for providing support for Chat protocols in *PickPacket*, Lastly the organizational flow of this report is explained.

A. Sniffers

Introduced in 1988 by Network General Corporation (now Network Associates incorporated); Sniffers are network monitoring tools also called as Network Analyzer, Network sniffers are software applications bundled with hardware devices and are used for eavesdropping on the network. Generally sniffers work by putting the Ethernet hardware (the standard network adapter) into promiscuous mode. A simple sniffer just writes all the packets in the network onto disk. The first level of filtering is based upon network parameters like IP addresses,

protocols and port numbers present in the packet. This level of filtering is generally supported at the kernel level also.
This tool can filter packets based on the rules formed using network parameters. Network Associates Incorporated [9] have a range of sniffers including VOIP sniffers.

B. PickPacket

It supports various levels of packet filtering mechanisms while sniffing the packets. It can capture packets based on the network level parameters like IP-addresses, port numbers and protocols. Two, it provides two modes of packet capturing, "PEN" mode and "FULL" mode.
PickPacket comprises of 4 components. *a. Configuration -File-Generator* is a JAVA based user interface for specifying the values of filtering parameters. *b. Filter* is an online component which selects the connections based on the criteria specified in configuration file, *c. PostProcessor* is an offline post-capture analysis tool that extracts per application protocol metadata information from the output file generated by *Filter* component, and *d. DataViewer* is a web based application to render the metadata generated by *PostProcessor* in a user interactive manner.

C. NEED FOR CHAT PROTOCOLS SUPPORT IN PICKPACKET

The aim of PickPacket is to concentrate on those application layer protocols which form significant portion of the Internet traffic and are used to communicate information among users. In this view, earlier implementation of PickPacket had support for four application protocols named FTP, HTTP, SMTP and Telnet. Chat protocols, by which a group of users form a network to communicate information among them, have gained popularity in the last few years. Chat protocols are generally used to establish collaboration among geographically distributed development teams. Active use of these protocols on the Internet motivated the need for support of chatting protocols in PickPacket,

As a step towards giving support for chatting protocols in PickPacket, we have considered two most popular protocols named IRC and Yahoo messenger, Internet Relay Chat (IRC) was one of the first chat protocols, and quickly gained the status of being the most popular one on the net, Yahoo messenger is another popular chat protocol which is proprietary. Our contribution includes extension of PickPacket for two chatting protocols (IRC and Yahoo Messenger). All components of the PickPacket have been upgraded for the support of new protocols, PickPacket has been tested for correctness and performance measurement.

II. PICKPACKET: ARCHITECTURE AND DESIGN

In this section, PickPacket architecture is discussed followed by a brief explanation of the design of each component in the architecture. Detailed documentation about the design and implementation of PickPacket can be found in [7].

A. Architecture of PickPacket

PickPacket can be logically viewed as a composition of four independent components working in a pipeline. These components are - *PickPacket Configuration File Generator* deployed on a Windows/Linux machine, *PickPacket Filter* deployed on a Linux machine, *PickPacket PostProcessor* deployed on a Linux machine and *PickPacket DataViewer* deployed on a Windows/Linux machine. Ideally, it is possible to run all the four components on a single machine. Pictorial view of PickPacket architecture can be seen in Figure 2,1, *Configuration File Generator*, the first component, is a JAVA based user interface that can take filtering criteria at different levels. It generates configuration files which are input to *Filter*. *Filter*, an online component, reads all the packets and stores those packets which match the criteria in the configuration file, *PostProcessor* is an off-line capture analysis tool that accepts output files of *Filter* and extracts meta information in a fixed directory structure. *Data Viewer* is a web based user interface to render the *Postprocessor* outputted metadata in an interactive fashion.

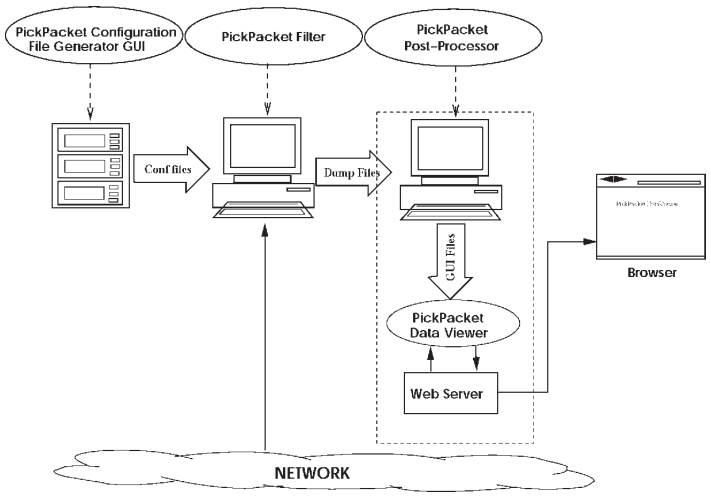


Figure 2.1: Architecture of PickPacket

B.1 Pickpacket Design

In this section the design of four components of PickPacket are separately analyzed.

B.1.1.Configuration File Generator

PickPacket Configuration File Generator generates two configuration files containing values of several parameters needed by *Filter*. One file stores filtering criteria given by the user. These criteria could be at different levels of protocol hierarchy. The other file contains values for different buffering parameters. It has been modified to generate two files so that a normal user is only concerned with filtering criteria. These files are written in HTML like syntax. An example configuration file set is given in *Appendix A.*

The configuration file containing filtering criteria has three sections: 1st section contains specification of the output files that would be created by Filter; 2nd section contains criteria for filtering packets based on source and destination IP addresses, transport layer protocol, and source and destination port numbers. It also maintains the application layer protocol filter to be used for the packets belonging each such criterion. This information is used to demultiplex packets to the correct application layer protocol filter and 3rd section contains specific criteria corresponding to an application layer protocol. Application layer content of the packet is investigated for match against the corresponding protocol criteria.

The configuration file storing system parameters values has two sections: the 1st section comprises of entries giving a value to the maximum number of connections the filter should monitor simultaneously for each application protocol and the 2nd section has one entry per application layer protocol specifying the number of history packets the filter should be able to keep while monitoring a connection of that protocol for criteria match. The values in these sections are used to pre-allocate memory buffers in Filter, 2.2.2 PickPacket Filter

B.1.2.The PickPacket Filter

Finally, application specific filtering reduces to text search in the application layer data content of the packets. This module is called the *TCP Connection Manager*. There is one such module for each application level filter that does the text string match in the application payload.

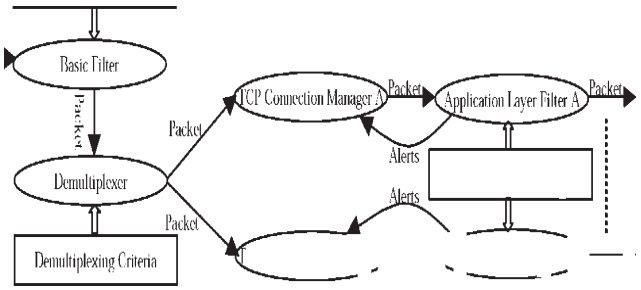


Figure 2.2: Filter Design

All modules in the design are represented by ovals in the figure, *Basic Filter* module works at the first level of filtering, while the *Application Layer Filters* work at the second and third levels of filtering. *Demultiplexer* module directs the packets to one of the application layer filters. There is a *TCP Connection Manager* for each application layer protocol. First the connection manager need not determine the sequecing of packets for all connections. Connection manager is anyway maintaining state for each connection till a criteria match for determining sequencing of packets. Alerts would also involve asking connection manager to either delete these packets or store them to disk.

A module *Initialize* has been added for initialization of all filtering modules based on the configuration file. *Demultiplexer* is provided the facility of calling *Output File Manager* directly so that the basic filter can directly store packets without resorting to application layer protocol based filtering, if necessary, *Connection Manager* can also directly store packets to the disk.

B.1.3PostProcessor

The PostProcessor is an offline analysis tool to extract the metadata corresponding to all the connections captured by the filter. Ideally PostProcessor should meet following object vies:

- **Session breaking:** A connection is identified by 4-tuple, There can be more than one connection existed at different time intervals but with same 4-tuple identifiers. Each of these connections is regarded as a session of corresponding 4-tuple, The filter output file contains packets belonging to several sessions. Before attempting to extract any data from these files, sessions need to be separated,
- **Metadata extraction:** Metadata includes important fields and entities present in the data content belonging to an application layer protocol. For example, it is email addresses and emails incase of SMTP, usernames and files incase of FTP, Metadata extraction from each session should be handled separately and should be stored in a fixed structure.

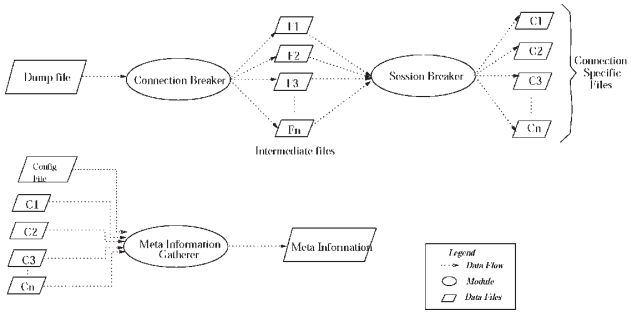


Figure 2.4:PostProcessor Design

Figure 2,4 shows the design of *PickPacket PostProcessor* which captures above two objeetvies. The first module, *Connection breaker*, accepts the filter outputfile as input and produces set of files. Each file contains all the packets belongs to a 4-tuple, It works by reading each packet from the input dumpfile and writing it to a specific file that is named with the 4-tuple of the packet. This module writes all the packets sent by either entity of the 4-tuple into a single file.

The second module *Session breaker* reads each file generated by *Connection breaker* and splits that file into as many number of files as the sessions involved in the connection corresponding to that file. If the file belongs to a UDP connection, *Session breaker* direelv produces same file as output. If the file belongs to a TCP connection, it uses a TCP like engine to identify the sessions involved in that connection. Then, it produces one file for each session where packets are sorted by their time. Packets with same time are sorted by their sequence numbers.

The third module, *Metainformation gatherer*, employs one metadata extractor for each application layer protocol. It reads one session file at a time and directs it to appropriate metadata extractor depending on the application protocol of the session. These metaextractors store information in a fixed format. This format accommodates one directory per session which contains all the details regarding that session stored into separate

files. File named "tepipinfo" contains summary information of the session that includes network parameter details, application layer protocol and list of matched keywords given as criteria. File named "appinfo" contains the meta data of the session that is specific to its application protocol. All the files are stored in standard INI format.

B.1.4 Data Viewer

PickPacket DataViewer is a web based application that is aimed to render the post- processed information in user interactive manner. It is written in PHP [11] script. 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,

PostProcessor output directory is placed in a fixed path that is known to DataViewer, At first, DataViewer lists all the directories present in that path. For any dumpfile directory selected by user, it lists summary of all the connections present in the directory. This summary includes the network parameters, application protocol and list of matched keywords. Connections can be sorted based on any one of summary fields. The DataViewer allows examining the details of a connection and can show the data for that connection through appropriate user agents.

The Data Viewer provides user with facilities like downloading the captured e-mails, viewing browsed web pages etc. The communication between connection entities can be seen in a separate dialogue window. The configuration file used for the filtering can be viewed. The connection-specific output file for each of the connection can also be downloaded separately. 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.

III. Adding Support for IRC: Design and Implementation

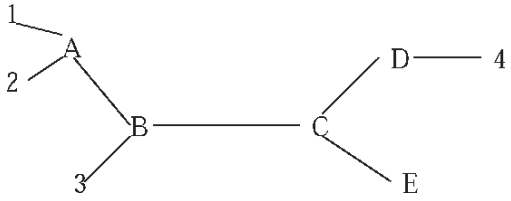
The Internet Relay Chat (IRC) [12] was one of the first chat protocols on the Internet, This chapter discusses the design and implementation issues in adding support for IRC in *PickPacket.* PickPacket requires one new module in each of its components inorder to support IRC, Section 3,1 gives brief overview of the IRC protocol. Following Section explains the design and implementation of *IRC Filter*, an application protocol filter module in *PickPacket Filter and* describes the design of *IRC Metahandler*, an application protocol metadata extractor in *PostProcessor*. It also describes *IRC Viewer* that has been added to DataViewer,

C.1 Internet Relay Chat (IRC) Protocol

The IRC has been designed over a number of years for use with text based conferencing, Following section explains the protocol in brief. This is followed by description of command sequences involved in the execution of the protocol,

C.1.1 Protocol Overview

A typical IRC setup consists of group of servers and clients. Servers form the backbone of IRC, providing points to which clients may connect to talk to each other, A server also forms a point for other servers to connect to, forming an IRC network. The only network topology allowed for IRC servers is that of a spanning tree, where each server Rets c\ central node for the rest of the net it sees. Figure 3,1 shows a sample configuration of IRC network.



Servers: A, B, C, D, E Clients: 1, 2, 3, 4

Fi**gure 3.1:** An example of small IRC network

The IRC network can become disjoint when a connection between two servers breaks. In such a situation, the channel in each partition only consists of those clients which are connected to servers in that partition. It is possible that in one of the partition there is no client belonging to a particular channel. In this case, the channel will cease to exist in that partition. When the connection is re-established, the connecting servers exchange membership and modes of all channels. If a channel exists on both partitions, the join messages and modes are interpreted in an inclusive manner,

C.1.2 Command Sequences

An IRC client can connect to any server by sending a "NICK" message that contains the nickname of user's choice. The receiving server broadcasts the arrival of new nickname to all other servers. If a "NICK" message arrives at a server which already knows about an identical nickname for another client, a nickname collision occurs. As a result of nickname collision, all instances of nickname are removed from the server's database and "KILL" command is issued to remove the nickname from database of all other servers. If the server reeieves an existing nickname from a client which is directly connected, it may issue a collision error to the client, and not generate any "KILL" commands. After "NICK" command, client sends "USER" command that is used to specify the username, hostname, and servername.
A client's nickname is a dynamic identity which can be changed at any time by resending "NICK" command,

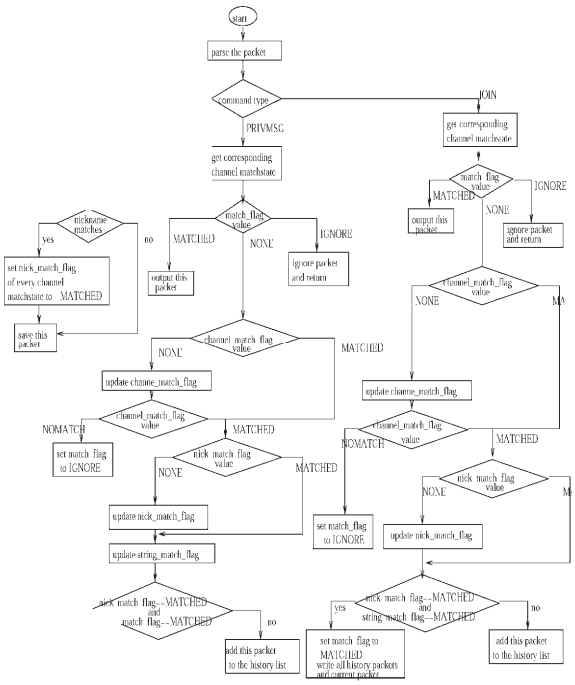


Figure 3.2 Working of IRC filter

A client can connect to any channel by sending a "JOIN" command that takes channel name as an argument. Whether a client is allowed to join a channel is checked only by the server to which the client is connected; all other servers automatically add the user to the channel when such a request is received from other servers. This allows each server to know where to find the users who are members of the channel. If "JOIN" is successful, the user is sent the channel's topic and the list of users on the channel, A channel operator is the user who joined the channel first. The operator of a channel can set the characteristics of a channel by using the "MODE" command. Any user on an invite-onlv mode channel can invite new members using "INVITE" command, A client can send a private message using the command "PRIVMSG", This command takes the receiver's name and the text to be sent as arguments. The receiver argument can be nickname of the receiver of the message. This can also be a list of names or channels separated with commas. The "PART" command causes the client sending the message to be removed from the list of active users for all given channels listed in the argument string,

A client session ends with a "QUIT" command. The server must close the connection to a client which sends the "QUIT" command. If a server wishes to break the connection to another server, it must send "SQUIT" command specifying the name of the other server as the parameter,

C.2 IRC Filters

The *IRC Filter* is an application protocol filter module in the component *PickPacket Filter*. This is meant for capturing selective data transferred in IRC protocol sessions, Section 3,2,1 gives the objectives of filter module. Section 3,2,2 discusses the design and **implementation** details of *IRC Filter*.

C.2.1 Goals

IRC channels are the entities of interest in an IRC session. Therefore, the filtering criteria for the *IRC Filter* should be able to specify details about a channel, A channel has a name, has members who are communicating with each other, and there is information that is being communicated. We should be able to filter based the values of channel names, nicknames, and keywords. Using channel names and nicknames, targeted monitoring can be done for the channels having certain name and members. The 'keyword' field can be used to specify the strings for which *IRC Filter* should try to match in the text messages of channels.

The *Configuration File Generator* component has been updated to provide interface for accepting the values of IRC criteria. Objective of the *IRC Filter* is to capture only those conversations which match channel name, member nicknames and message strings.

C.2.2 Design and Implementation

First flag, 'channel^match^flag', is set to "MATCHED" when the channel's name matches with one of channel names present in the criteria. If no channel name is given in the filtering rules, this flag is set to "MATCHED" for all channels. Second flag, 'nick^match^flag', is set to "MATCHED" when nickname of one of the members of the channel matches with one of the nicknames present in the criteria. Fourth flag, 'match^flag', is set to "MATCHED" when the channel has above three flags set to "MATCHED". The *IRC Filter* receives packets from the *TCP Channel Manager* module which also gives packet's connection information. If the command type is "NICK", the filter stores the nickname present in the packet. If it matches, 'niek^mateh^flag' of every channel data structure is set to "MATCHED".

If the command type is "PRIVMSG", the filter extracts channel name, sender's nickname and message present in the packet. The filter considers data structure corresponding to the channel name. If the 'mateh^flag' is set to "IGNORE", this packet is ignored. Then the filter considers 'string^mateh^flag'. If all three flags ('channel^match^flag','niek^mateh^flag' & 'string^mateh^flag') are set to "MATCHED", the filter sets the 'match^flag' to "MATCHED" and writes packets present in the history list including the current packet. Otherwise, the filter remembers current packet in the history list of this channel.

If the command type is "JOIN", the filter extracts channel name and new member's nickname present in the packet. The filter considers data structure corresponding to the channel name. If the match state has 'match^flag' set to "IGNORE", this packet is ignored. If the 'mateh^flag' is set to "MATCHED", the filter writes this packet to outputfile. If the 'mateh^flag' is set to "NONE", the 'niek^mateh^flag' is considered, The filter tries to match the new member nickname with list of nicknames present in the criteria and sets 'niek^mateh^flag' to "MATCHED" if nickname matches. If the flag is set to "MATCHED" and the value of 'string^match^flag' is "MATCHED", the filter sets the 'mateh^flag' to "MATCHED" and outputs packets present in the history list including the current packet. In the "PEN" mode of packet capturing, the filter writes only the first packet of any channel that has matched the criteria.

C.3 IRC Metahandler and Viewer

The *Metahandler* maintains a table where each entry contains the name of the file corresponding to a channel along with list of matched nicknames and keywords in the IRC criteria. It parses each packet read from the session file to figure out the packet's command type, just like the way filter did. If the packet belongs to "USER" or "NICK" command, it stores the command argument. If it belongs to "PRIVMSG" or "JOIN" type, then it looks up the table to find the name of its file. If no entry is found in the table, then the handler would enter new entry into the table with a new file name. Therefore all the messages corresponding to a channel would be written into a separate file in a INI format. These messages are checked for updating the list of matched nicknames and keywords in IRC criteria.

The *IRC Viewer* is a module in *PickPacket DataViewer*. It is used to display the connections belonging to IRC protocol. This module uses the "ircinfo" file to display the channels present in a connection. It shows the

channel conversations in a separate dialog box. User can search among the existing IRC sessions with different parameters.

IV. Adding Support for Yahoo Messenger: Design and Implementation

D.1 Yahoo Messenger Protocol

Yahoo Messenger is a proprietary protocol used to provide instant messaging and chatting services among yahoo registered users only. As there is no official documentation available about this protocol on the net, we have taken the help of an unofficial documentation [14]. We have also analyzed at the source code of Gaim [4], an open source multi-protocol instant messaging client supports Yahoo messenger protocol, to draw more information about the protocol. We have also conducted lab experiments to monitor protocol conversations and to verify its accordance with the protocol specifications that we have determined.

D.1.1 Protocol Overview

The server host authenticates client's hash response by sending a reply message. Now, the user can communicate with the other users connected to the server through Instant Messages(IM) or Chatrooms, Instant Messaging allows the user to exchange messages with any other user in real time. If the destination user is already connected to the server, the server will immediately dispatch that message to him. The server checks uniqueness of the user's chatroom name and sends its acceptance reply to the client. Once the chatroom is established, messages sent by any member will first reach the server.

D.1.2 Command Sequences

Yahoo Messenger protocol classifies the commands involved in the protocol into several service types where each service type represents particular state of the protocol execution. All the commands follow a particular format shown in Figure 4,1, This format consists of two fields: a fixed-length *header* and a variable length *data*. The *header* field is a group of six fixed-length subfields. First subfield is always a string "YMSG", The second subfield is the version of the protocol it is using. Third subfield gives the length of the *data* field. Fourth subfield specifies the service type of this command. Fifth and sixth subfields are speefie to the user. They give the status of user and his id respectively. The *data* field is a collection of attribute-value pairs separated by a fixed two-bvte separator.

The server uses *challenge-response* mechanism to authenticate its users. During the authentication phase of the protocol, both the server and client communicate with commands of service type "AUTH". The client uses this service type to send its Yahoo username after it has established a TCP connection with the server. Then the server challenges with a random number. The client responds with an MD5 hash of the Yahoo user's password using the random number sent by the server. The server authenticates client's response after checking whether the user is already logged in or not.

For transferring Instant Messages (IMs), commands of service type "MESSAGE" are used. Fields in these commands include the receiver’s username and the message. to be sent. While initiating new chat sessions, commands of service type "CHAT- LOGON" are used. These commands allow the user to give his choice of chatroom name. The client uses commands of service type "CHATMSG" for sending messages to chatrooms. If a new user joins the chatroom, a command of service type "CHATJOIN" with the identity of the new user would be sent by the server to all the existing members of the chatroom. Similarly, a command of service type "CHATLEAVE" would be sent if any user leaves the chatroom.

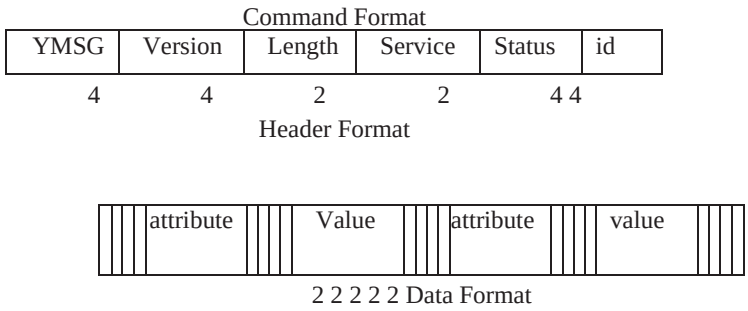


Figure 4.1: Yahoo Command Prompt

D.2.1.Objectives

Entities of interest in a Yahoo session are IMs and chatrooms. Therefore, the filtering criteria for *Yahoo Filter* should be able to specify details about any IM or Chatroom, We have decided that filtering should be possible on the basis of user- names, chatroom names, vahoo-ids and keywords, Usernames are the identities with which a user tries to authenticate himself to the server. Using chatroom names and yahoo-ids, targeted monitoring can be done for chatroom sessions having certain name and members. Keywords are the strings for which the filter should try to find a match in both chatrooms and IM messages.

The *Configuration File Generator* has been updated to provide an interface for accepting the values of Yahoo criteria. The objective of *Yahoo Filter* is to capture only those conversations of chatrooms and IMs which have the name, members and message strings matching the criteria,

D.2.2. Design and Implementation

First flag, 'chatroom^match^flag', is set to "MATCHED" when the chatroom's name matches with one of chatroom names present in the criteria. If the chatroom name doesn't match, the filter sets the 'ehatroom^mateh^flag' to "NOMATCH", and the 'mateh^flag' to "IGNORE". If it matches, the £yahoo-id_match_flag' is considered. Then the filter considers 'string^match^flag'. If the matchstate has 'chatroom^match^flag',
£yahoo-id_match_flag' and £string_match_flag' set to "MATCHED", the filter sets the £match_flag' to "MATCHED" and writes packets present in the history list including the current packet. The filter considers matchstate corresponding to the IM, If the matchstate has £match_flag' set to "IGNORE", this packet is ignored. If the £match_flag' is set to "MATCHED", this packet is written to the disk. If the matchstate has £yahoo-id_match_flag' and £string_match_flag' set to "MATCHED", the filter sets the £match_flag' to "MATCHED" and writes packets present in the history list including the current packet. Otherwise, the filter remembers current packet in the history list of match- state.
If the £mateh_flag' is set to "MATCHED", the filter writes this packet to outputfile. If it doesn't match, the filter sets 'ehatroom^mateh^flag' to "NOMATCH", and 'mateh^flag' to "IGNORE". Otherwise, the filter considers £yahoo-id_mateh_flag'. The filter tries to match the new member's yahoo-id with list of vahoo-ids present in the criteria and sets 'yahoo- id _ match _ flag' to "MATCHED", if yahoo-id matches. If the flag is set to "MATCHED" and the value of £string_match_flag' is "MATCHED", the filter sets the 'match^flag' The filter sets the 'match^flag' to IGNORE to ignore future packets.

D.3 Yahoo Metahandler and Viewer

It extracts meta data from the session files belonging to Yahoo Messenger protocol. This module writes messages belonging to each chatroom or IM in a separate file. The Metahandler generates one "yahooinfo" file for each session. The *Metahandler* maintains a table where each entry contains the name of the file corresponding to a chatroom or IM along with the list of matched keywords in the Yahoo criteria.

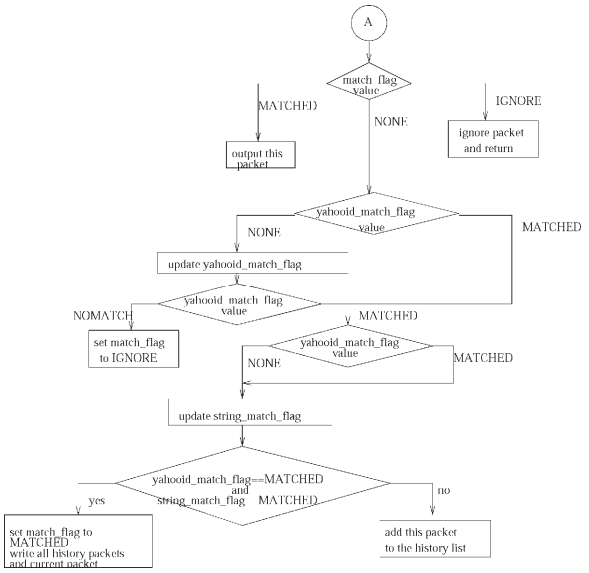


Figure 4.2: Working of Yahoo Filter

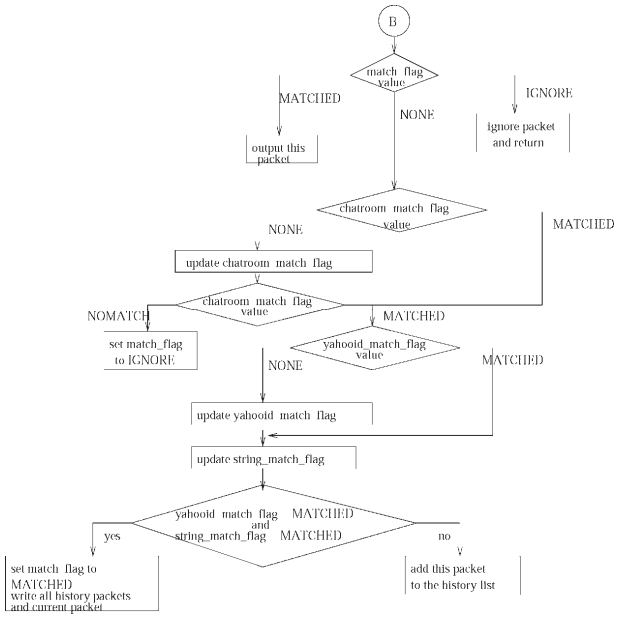


Figure 4.3: Format of yahooinfo file

V. TESTS AND RESULTS

In this chapter, we explain the experiments conducted to test the *PickPacket* with the new extensions in an actual network. The experiments are aimed to test the correctness and performance of the newly added modules for IRC and Yahoo Messenger. The previous dealt with the correctness testing. The behavior of the protocol filter has been tested in both "PEN" and "FULL" modes of capturing. The experiments for determining performance of the application layer filters are similar to experiments described in [10].

E.1.Correctness Verification

Goal of this experimentation is to test whether the filter and other components are working correctly i.e. the packets stored on the disk must correspond to a session which has matched a criteria mentioned in the configuration file, and one should be able to post-process and view such connection. An application protocol criterion contains values for several fields. A user can choose to store values for some of the these fields, A typical criteria may contain values for no fields which will cause the filter to write all the packets belonging to the corresponding protocol.

E.1.1.IRC Filter

An IRC criteria contains values for three fields - *channelname, nickname* and *keyword.* We have created a test configuration file that contains seven possible IRC criterion. It is observed that *IRC Filter* stored those sessions which strictly matched the criteria. In case of "PEN" mode of packet captruing, the filter stored only the first packet of the channels matching the criteria,

E.1.2.Yahoo Filter

An Yahoo criteria contains values for four fields - *username, charoomname, yahoo-id* and *keyword.* There are fifteen possible ways of preparing a criteria that contains values for atleast one field, *Yahoo server* in Yahoo Messenger protocol communicates over HTTP protocol with the clients standing behind proxy servers. As HTTP protocol uses separate TCP connection for every message that is transfered between server and client, *Yahoo server* communication with these clients involves numerous HTTP connections. It is observed that *Yahoo Filter* stored only those sessions which strictly match the criteria.

E.2.Performance Evaluation

Each packet received by the *PickPacket Filter*, is processed by the application protocol filter for criteria match. If the protocol filter is slow in filtering packets, the kernel may start dropping packets when its internal buffers gets filled up. The earlier version of *PickPacket Filter* works at line speed in case of 100Mbps Ethernet segment. The other instance of *PickPacket Filter* with application layer protocol specific criteria was run on second machine. The output file for this filter was a normal file. For simplicity the former packet filter is referred to as *counting sniffer* while the other as *filtering sniffer*. In our experiment, *counting sniffer* processed 3533100 packets and *filtering sniffer* processed 3527900 packets.

The small difference in number of packets is due to delay while starting the sniffers manually. Thus the PickPacket Filter can work at line speed without loss of information. We have measured the usage of processor and memory while running the filter and found that they were less than 50% used. Therefore, the filter can run at higher traffic speeds.

VI. CONCLUSION

PickPacket is a useful tool for gathering and rendering information flowing across the network. PickPacket is architecturally divided into four components the *PickPacket Configuration File Generator*, the *PickPacket Filter*, the *PickPacket Post Processor*, and the *PickPacket Data Viewer*. Design of each of these components were briefly discussed, PickPacket uses in-kernel filtering to capture packets at the network level. The packets filtered by the in-kernel filter are passed to the application level filter for further processing.

Modules for filtering IRC and Yahoo Messenger protocol packets have been further discussed in this thesis. Users of PickPacket can specify names of channels, nicknames and text search strings for filtering packets belonging to IRC sessions, Usernames, chatroom names, yahoo-ids and keywords can be specified for filtering packets belonging to Yahoo Messenger sessions.

VII. FUTURE SCOPE

PickPacket currently supports SMTP, POP, IMAP, Telnet, FTP, HTTP, IRC, and Yahoo Messenger application level protocols. There is always scope for extending PickPacket to support other application level protocols, PickPacket doesn't have support for decompressing the compressed data to do string matching. This would be required as electronic transfer of data in compressed format is popular.

Due to recent concerns over the impending depletion of the current pool of Internet addresses and the desire to provide additional functionality for modern devices, a new version of Internet Protocol (IP) called IPv6 [5] is in the process of standardization. This version resolves unanticipated IPv4 design issues and is poised to take the Internet into the 21st Century, Therefore, PickPacket would need changes for compatibility with IPv6.

REFERENCES

[1] Loris Degioanni, Fulvio Eisso, and Piero Viano, "Windump". http://netgroup- serv.polito.it/windump,

[2] erald Combs et al. "Ethereal". Available at http://www.ethereal.eom.

[3] "Etherpeek nx". http://www.wildpaekets.eom.

[4] "Gaim:A multi-protocol instant messaging (im) client", "http: //gaim.sourceforge.net/".

[5] "Ipv6: The Next Generation Internet!", "http://www.ipv6.org".

[6] Van Jacobson, Craig Leres, and Steven MeCanne, "tepdump : A Network Monitoring and Packet Capturing Tool". Available via anonymous FTP from ftp://ftp.ee.lbl.gov and www.tcpdump.org.

[7] Neeraj Kapoor. "Design and Implementation of a Network Monitoring Tool". Technical report, Department of Computer Science & Engineering, IIT Kanpur, Apr 2001.

http://www.cse.iitk.ac.in/research/mtech2000/Y011111.html.

[8] Steve MeCanne and Van Jacobson. "The BSD Packet Filter: A New Architecture for User-level Packet Capture". In *Proceedings of USENIX Winter Conference*, pages 259-269, San Diego, California, Jan 1993.

[9] "Network Associates Incorporated", http://www.sniffer.com.

[1] Brajesh Pande. "The Nework Monitoring Tool - Pickpacket : Filtering ftp and http packets". Technical report, Department of Computer Science and Engineering, IIT Kanpur, Sep 2002, http://www.cse.iitk.ac.in/research/mtech2000/Y011104.ps.gz.

[2] "Php Site", http://www.php.net,

[3] J. Oikarinen D. Reed. "Internet Relay Chat Protocol". Technical report, 1993. http://www.faqs.org/rfcs/rfcl459.html.

[4] Stephen P. Smith, Henry Perrit Jr., Harold Krent, Stephen Meneik, J. Allen Crider, Mengfen Shvong, and Larry L. Reynolds. "Independent Technical Review of the Carnivore System". Technical report, IIT Research Institute, Nov 2000. http://www.usdoj.gov/jmd/publications/carniv_entrv.htm.

[5] Venkat, "Yahoo Messenger Protocol (unofficial doeumnetation)", "http://www.venkydude.com/articles/vahoo.htm".