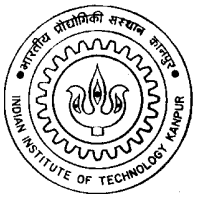
Implementation of RADIUS Support in PickPacket

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Sanjay Kumar Jain



to the

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

May, 2003

Certificate

This is to certify that the work contained in the thesis entitled "*Implementation of RADIUS Support in PickPacket*", by *Sanjay Kumar Jain*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

May, 2003

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

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

Abstract

The extensive use of computers and networks for exchange of information has also had ramifications on the growth and spread of crime through their use. Law enforcement agencies need to keep up with the emerging trends in these areas for crime detection and prevention. Among the several needs of such agencies are the need to monitor, detect and analyze undesirable network traffic. However, the monitoring, detecting, and analysis of this traffic may be against the goal of maintaining privacy of individuals whose network communications are being monitored.

PickPacket — a network monitoring tool that can handle the conflicting issues of network monitoring and privacy through its judicious use, is discussed in References [10, 15, 1]. This thesis discusses the implementation of RADIUS [24, 22, 23] support in PickPacket and how the information in RADIUS packets can be used to monitor dialup users who are generally allocated dynamic IP addresses by the Internet Service Provider.

Acknowledgments

I take this opportunity to express my sincere gratitude towards my thesis supervisors Dr. Dheeraj Sanghi and Dr. Deepak Gupta for their invaluable guidance. It would have never been possible for me to take this project to completion without their innovative ideas and encouragement. I also thank the other team members involved with the development of PickPacket - Neeraj, Brajeshji, Prashant, Abhay, Nitin, Ankit, Sachin, JVR Murthy and Srikanth for their cooperation and support. Abhay, Nitin and Ankit painstakingly performed several testings on PickPacket. Prashant always helped me whenever there was any problem related with linux system. I will also remember him for his small trainning on CVS. Sachin, Kanth and Murthy helped me in second release of PickPacket. I fondly remember Neeraj who explained me the various design issues of PickPacket. Unfortunately, he is no longer with us. I would like to thank BrajeshJi, for being cooperative. Without his constant support and ideas even after completion of his M.Tech., this work would have been dificult for me.

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 also wish to thank the Head of Computer Center, IIT Kanpur and and Navpreet SinghJi for loaning me Remote Access Server and Modem for testing the application. I also wish to thank MishraJi, Nadeem, Chandan, Anamika madam, Sandeep who were always very cooperative.

I would like to thank whole of mtech2001 batch for the times I shared with them. I always felt with them as if I am also of same age group.

I thank to Ministry of Communication and Information Technology, New Delhi for sponsoring this project.

I would like to thank my parents for taking me to this stage. It was the their blessing and support of my wife Jyoti which gave me confidence to do M.Tech. after a long gap of 7-8 years.

Finally, I thank my both daughters Prachi and Astha. It were the long hours stolen from the time due to them that make the story of my M.Tech.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The use of computers has rapidly increased in the last few decades. Computers can now exchange large volumes of information very fast. Coupled with this has been the exponential growth of the Internet. The Internet in all its various forms (the World Wide Web, email, chatrooms and many others) has opened up a whole new world to millions of us. Unfortunately, criminals have been just as quick to exploit its possibilities. They are increasingly relying on the net for communication and exchange of information pertaining to unlawful activity. Consequently the ability of law enforcement agencies to conduct lawful monitoring of the data flowing across the net can help detect and prevent crime. Such monitoring tools, therefore, have an important role in intelligence gathering. Companies can also use such tools to safeguard their information repositories and research efforts, in addition to preventing abuse of network facilities by employees. Thus there is a pressing need to monitor, detect and analyze undesirable network traffic.

However, the monitoring, detecting, and analysis of this traffic may be opposed to the goals of maintaining the privacy of individuals whose network communications are being monitored. PickPacket is a network monitoring tool that can address the conflicting issues of network monitoring and privacy through its judicious use. This tool has been developed as a part of the research project sponsored by the Ministry of Communication and Information Technology, New Delhi. The basic framework for this tool and design and implementation of application layer filter for Simple Mail

Transfer Protocol (SMTP) [11] and Telnet [18] has been discussed in Reference [10]. The design and implementation of application layer filter for Hyper Text Transfer Protocol (HTTP) [6] and File Transfer Protocol (FTP) [19] has been discussed in Reference [15]. The design and implementation of text string search in MIME-Encoded data has also been discussed in Reference [1]. This thesis discusses the design and implementation of application layer filter for the Remote Authentication Dial In User Service (RADIUS) Protocol [24, 22, 23].

1.1 Sniffers

The word "sniffer" is a registered trademark of Network Associates referring to the "Sniffer(r) Network Analyzer", a product introduced by them in 1988. The term 'sniffer' is more popular in everyday usage than alternatives like "protocol analyzer" or "network analyzer". Sniffers can be used both for legitimate network management functions and for stealing information off a network. Recently, sniffers have also found use with law enforcement agencies for gathering intelligence and helping in crime prevention and detection.

The primary mechanism of sniffing in ethernet is by putting the ethernet hardware into "promiscuous mode". Ethernet was built around a "shared" principle: all machines on a local network share the same wire. This implies that all machines are able to "see" all the traffic on the same wire. Ethernet card (the standard network interface card) is hard-wired with a particular MAC address and is always listening for packets on its interface. When it sees a packet whose MAC address matches either its own address or the link layer broadcast address (i.e., FF:FF:FF:FF:FF:FF for Ethernet) it starts reading it into memory. It rejects all packets whose destination MAC addresses are different from that of the card. But, it is possible to turn off this filtering mechanism of the card and collect all the frames flowing through the network, independent of their MAC address. This is known as putting the card into promiscuous mode. Sniffers put the network card in promiscuous mode.

A simple sniffer that just captures all the data flowing across the network and dumps it to the disk soon fills up the entire disk especially if placed on busy segments

of the network. Analysis of this data for different protocols and connections takes considerable time and resources. The privacy of individuals who are accessing and dispensing data which is not of user's interest is also compromised as all packets are being captured. It is therefore necessary to filter, on-line, the data gathered by the "promiscuous" network adapter.

There are three levels of filtering that can be applied on packets flowing across the network. The first level of filtering is based upon network parameters like IP addresses, protocols and port numbers. This level of filtering is generally supported by the kernel also. With in-kernel filtering several packets are rejected by the kernel itself and the overhead of copying these packets to application address space is avoided. This speeds up the filtering process. The second level of filtering is based on criteria specific to an application such as email-ids for the SMTP, user names for RADIUS etc. Since there is no support in the kernel for handling these parameters a user level application handles such filtering. The third level of filtering is based on the content present in the application pay load. For instance it may be desired to search for the presence of a text string in an e-mail sent during a SMTP session. Such filtering also needs to be handled by the user level application.

Sniffers dump captured data onto disk directly without any processing of this data. As such, this dump is not human-readable. Sniffers therefore come bundled with their own post-capture analysis and processing tools which extract information from the dump and present it in a human-readable form. In addition to just presenting the sniffed data, packet analyzers can be configured to provide different kinds of functionality like alerting network administrators if something has gone amiss.

Several commercially and freely available sniffers exist currently. Sniffers come in different flavors and capabilities for different Operating Systems. Ethereal [5] and WinDump [3] are two such popular tools for Windows. On UNIX sniffers are generally based upon libpcap and/or BPF [13] (Berkeley Packet Filter). Two popular sniffer tools on Unix are tcpdump [9] and Ethereal [5]. WinDump is a version of tcpdump for Windows that uses a libpcap-compatible library called WinCap.

Carnivore [25, 7, 8] is a tool developed by the FBI. It can be thought of as a tool with the sole purpose of directed surveillance. This tool can capture packets based

on a wide range of application-layer level based criteria. It functions through wire-taps across gateways and ISPs. Carnivore is also capable of monitoring dynamic IP address based networks. The capabilities of string searches in application-level content seems limited in this package. It can only capture email messages to and from a specific user's account and all network traffic to and from a specific user or IP address. It can also capture headers for various protocols.

1.2 PickPacket

PickPacket, the focus of this thesis and also discussed in Reference [10, 15, 1] is a monitoring tool similar to Carnivore. PickPacket can filter packets based on IP and TCP/UDP level criteria as well as application level criteria for several application level protocols such as FTP, HTTP, SMTP and Telnet. It also supports real-time searching for text string in application and packet content.

In this work, we have added support for the RADIUS protocol to PickPacket. RADIUS is a protocol that is commonly used to authenticate users dialing into a network. Such users are usually assigned IP addresses dynamically using the Dynamic Host Configuration Protocol (DHCP) [4]. Adding RADIUS support to PickPacket allows monitoring of the activites of a user whose RADIUS login name is known.

1.3 Organization of the Report

This thesis focuses in detail on filtering RADIUS data packets and using information in these packets to track dialup users. Chapter 2 describes the high level design and architecture of PickPacket. Chapter 3 briefly discusses the RADIUS protocol. Chapter 4 describe the design and implementation of the RADIUS filter, the post-processing details of the captured RADIUS packets and the user inteface provided for viewing RADIUS packets information. Chapter 5 describes the setup used for testing the filter. The final chapter concludes the thesis with suggestions for further work.

Chapter 2

PickPacket: Architecture and Design

This chapter discusses the architecture and design of PickPacket and details of the packet filtering component in PickPacket. First the architecture of PickPacket is discussed and its various components are identified. Discussion on these components is then undertaken with a view to elaborate on the design of the filtering mechanisms in PickPacket. Detailed design and implementation details are discussed in Reference [10].

2.1 The Architecture of PickPacket

PickPacket can be viewed as an aggregate of four components ideally deployed on four different machines. These components are – the *PickPacket Configuration File Generator*, a JAVA GUI deployed on a Windows/Linux machine; the *PickPacket Filter*, deployed on a Linux machine; the *PickPacket Post Processor*, deployed on a Linux machine; and the *PickPacket Data Viewer*, GUI deployed on a Windows machine. An architectural view of PickPacket is shown in Figure 2.1 where these components are shown in rectangles.

The PickPacket Configuration File Generator is used first to specify the criteria for capturing the packets. The criteria specified by the user are saved in a file which is called the configuration file. This configuration file needs to be transferred to the machine where the PickPacket Filter would run. The PickPacket Filter captures

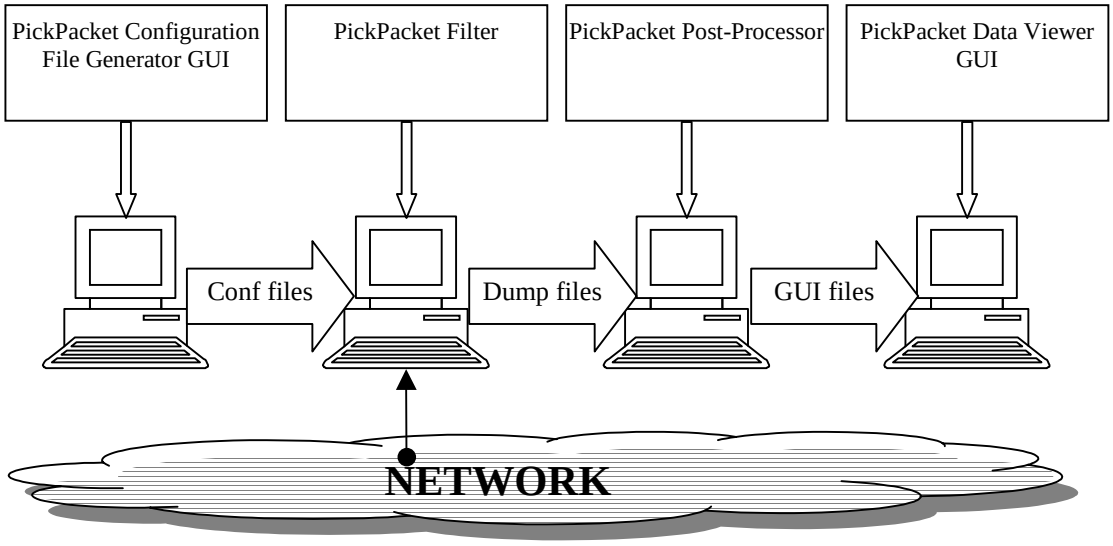


Figure 2.1: The Architecture of PickPacket [15]

packets according to the criteria specified in the configuration file and stores them to some storage device. Then the file containing the dump packets is processed offline using the PickPacket Post-Processor. The PickPacket Post Processor would typically run on some machine other than the one on which the PickPacket Filter is running. The Post Processor processes the captured data and retrieves the meta information. The meta information now is transferred to the machine running PickPacket Data Viewer. The Data Viewer reads this meta information and displays it to the user.

2.2 The PickPacket Configuration File Generator

The PickPacket Configuration File Generator is a Java based graphical user interface (GUI) that is used for specifying the rules for capturing the packets. The rules are then saved in a file. This file which is called configuration file is input to PickPacket Filter . This file is a text file with HTML like tags. A sample configuration file is given in *Appendix A*. This file has four sections:

1. The first section contains specifications of the output files that are created by the PickPacket Filter for storing captured packets. There is no restriction on number of output files. The last file can have a size of "0" meaning potentially infinite size. A feature in the configuration file is the support for different output file managers. This feature would be useful if captured packets have to be stored in formats other than the default pcap [26] style format.

2. The second section contains criteria for filtering packets based on source and destination IP addresses, transport layer protocol, and source and destination port numbers. The application layer protocol that handles packets that match the specified criteria is also indicated. This information is required for demultiplexing packets to the correct application layer protocol filter.

3. The third section specifies the maximum number of simultaneous connections that can be monitored for any application. This is used for memory allocations. The default value set by the configuration file generator is 500 for each application protocol.

4. The fourth section comprises of multiple subsections, each of which contains criteria corresponding to an application layer protocol. Based on these criteria the application layer data content of the packets are analyzed. Filtering criteria for SMTP, HTTP, FTP and Telnet can be specified in these subsections. The application layer protocol subsection also specifies the mode of operation of the filter("PEN" or "FULL") for this application layer protocol.

2.3 PickPacket Packet Filter: Basic Design

The PickPacket Packet Filter reads packets from the network by putting the network interface card into promiscuous mode. The packets which matches the criteria specified by the user are stored in a file for further analysis. This section presents the design of the PickPacket Filter.

The PickPacket filter can filter packets at three levels.

1. Network level(IP addresses, port numbers, etc.).

2. Application level(user names,email-ids, filename etc.).

3. Application level content(text strings).

The first level of filtering has been made very efficient through the use of in-kernel filters [13], as only packet which matches the network level criteria are copied from kernel space to user space. Since the content of application can be best deciphered by

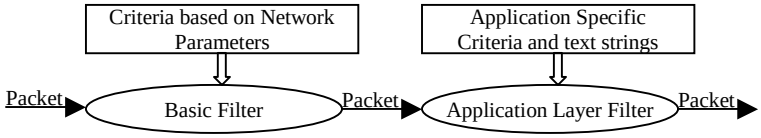


Figure 2.2: Filtering Levels [15]

the application itself, the second and third levels of filtering are combined. Figure 2.2 illustrates the various levels of filtering. The Basic Filter takes network parameters based criteria as input, and reads packets from the h/w interface and passes those packets that matches the specification criteria to the next level filter. All other packets are dropped. The Application Layer Filter takes as input the application specific filtering criteria , and packets passed from the Basic Filter. The detailed design of the PickPacket filter is shown in Figure 2.3.

The PickPacket Filter has separate filters for different application layer protocols. Thus there is an SMTP Packet Filter for filtering SMTP packets, an HTTP Packet Filter for filtering HTTP packets etc. This design has the advantage that it is easy to enhance the capability of the filter by adding new application layer protocol filters. A demultiplexer is provided between the basic filter and application layer filters. The demultiplexer decides which application layer filter should get the packet for the next level of filtering. The demultiplexer uses its own set of criteria for demultiplexing packets.

The application layer filter which gets the packet checks for application specific criteria (email-id, username etc.). Finally, application specific filtering reduces to text search in the application layer data content of the packets. In case of communications over connection oriented protocol, the text search handles the case where

the desired text is split across two or more packets. It also handles the case where packets are received out of sequence. To know whether a packet is out of sequence a component is provided between demultiplxer and application layer filter. This component is called the *TCP Connection Manager*. This component is used by all application level filtering modules that allow searching for text strings in the application pay load.

The TCP connection manager is designed in such a way that it needs to determine the sequencing for only those connections that an application layer filter is interested in. For this reason, it provides functions by which an application layer protocol filter can alert it so as to maintain the sequence information for a connection.

The connection manager maintains a list of state information for all the connections on which it has been alerted. This information is separately maintained for every connection, hence retrieving requires searching the list based on the four tuples (source IP, destination IP, source port and destination port). The connection manager also maintains a reference to the data that the application layer protocol filter has built for a particular connection. On receiving a packet the connection manager searches its list and retrieves this data and the reference to the data required by the protocol filter for that connection. After processing the packet it passes on the packet and the reference to the data required by the protocol filter. Whenever the application layer protocol filter wants to maintain state information about a connection, it passes this information to the connection manager by alerting it.

The module *Initialize* is used for initializations dependent on the configuration file. Another module, the *Output File Manager*, is responsible for dumping filtered packets to the disk. The *Filter Generator* module is used for generating the in-kernel BPF code. Hooks are provided for changing the BPF code on-the-fly. Functions that can generate the filter code based on changed parameters can be called by application level filters such as FTP during "PASSIVE" mode of file transfers. The *Demultiplexer* can also call the *Output File Manager* directly so that the filter can directly dump packets without resorting to application layer protocol based filtering, if necessary. The *Connection Manager* can also directly dump packets to the disk. This is required

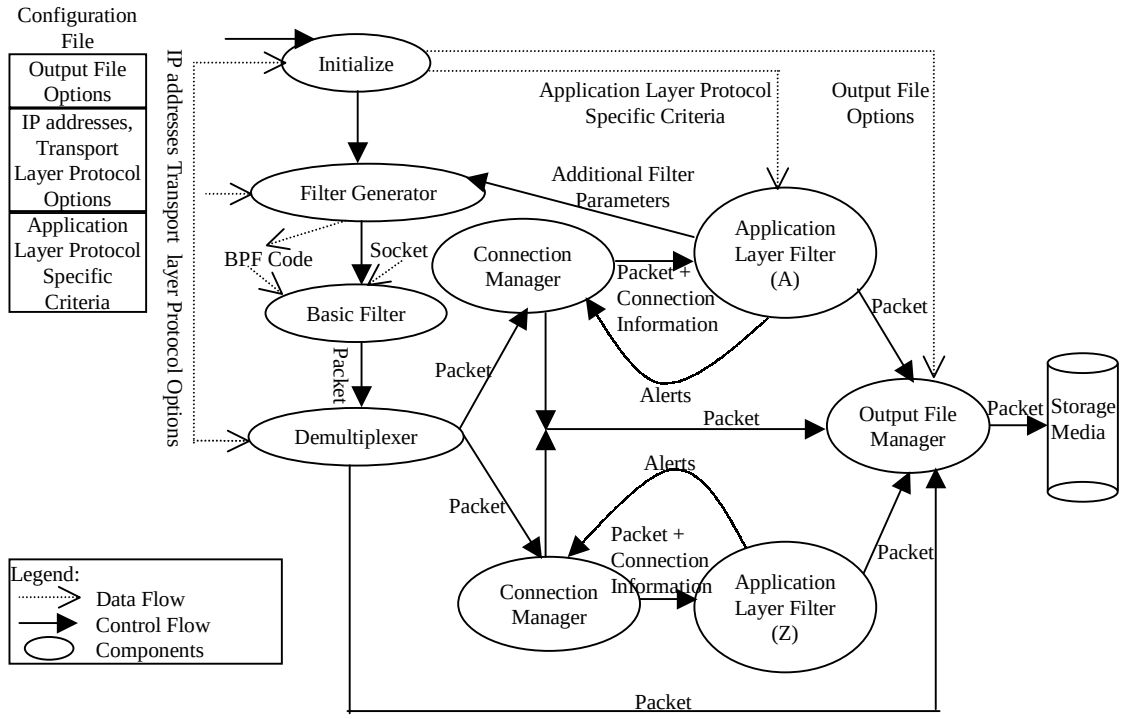


Figure 2.3: The Basic Design of the PickPacket Filter [10]

when all criteria have matched for a specific connection and the connection is still open. More details of these components can be found in Reference [10].

2.3.1 PickPacket Filter: Output File Formats

The *output file manager* can store files in any format. However, the output file manager provided by PickPacket stores output files in the *pcap* [26] file format. This file starts with a 24 byte pcap file header that contains information related to version of pcap and the network from which the file was 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 of the packet when it was read from the network, the length of the packet when it was saved and the time at which the packet was read from the network. The data stored in pcap file format

can also be viewed using utilities like tcpdump etc. This standard format also allow us to use some other tool for analysis of captured data.

2.3.2 PickPacket Filter: Text String Search

The PickPacket Filter contains a text string search library. This library is extensively used by application layer filters in PickPacket. This library uses the *Boyer-Moore* [20] string-matching algorithm for searching text strings. This algorithm can be used for both case sensitive and case insensitive search for text strings in packet data.

2.4 The PickPacket Post-Processor

The packet filter writes filtered packets to an output file that is analyzed offline by the PickPacket Post-Processor. This processing includes separating packets based on the transport layer protocol and the application layer protocol. The detailed description of Post Processor is given in Reference [10].

The Post Processor has three components – the *Sorter*, the *Connection Breaker*, and the *Meta Information Gatherer*. These are shown in Figure 2.4.

The packets present in the output file may not be in the order they were transmitted on the network. Therefore the *Sorter* module is used to sort the packets present in the output file generated by the packet filter based on the time stamp value corresponding to the time the packets were read off the network. The *Connection Breaker* module reads the sorted output file and retrieves the connection information from the packets belonging to a connection oriented protocol and separates them into different files. Internally connection breaking is accomplished by a TCP state machine [17] based process. Packets belonging to a connectionless protocol like UDP [16] are separated based on the communication tuple. The *Meta Information Gathering Module* reads these connection specific files and retrieves the meta-information of every connection. Each application requires different meta-information and packets belonging to a particular application are processed by meta-information gathering modules for that application. The meta-information of application layer protocols

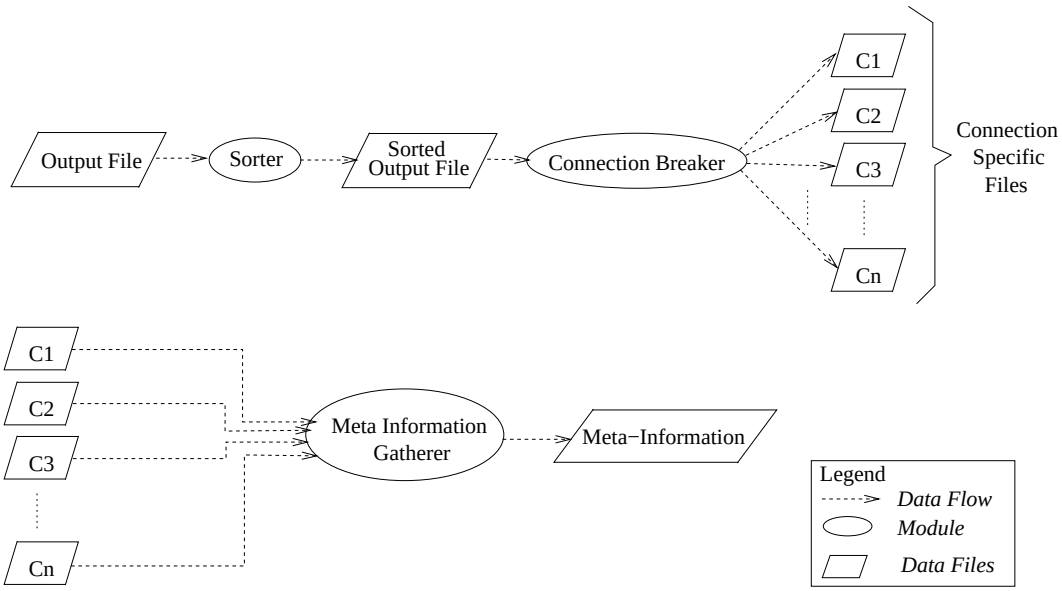


Figure 2.4: Post-Processing Design [10]

includes important fields present in the data content such as e-mail addresses for SMTP connections, usernames for FTP connections, URLs for HTTP etc. The meta-information for different application layer protocols is stored separately.

2.5 The PickPacket Data Viewer

The PickPacket Data Viewer is used for rendering the post-processed information. This is a Visual Basic based GUI and runs on Windows. The choice of this platform was made for rapid prototyping and the rich API (Application Program Interface) library that is provided in Windows for rendering content belonging to an application. The Data Viewer reads the meta-information files and lists all connections by application type, source and destination IP addresses, and other such fields based on the meta-information that has been provided by the Post-Processor. These connections can be sorted and searched based on these fields. The Data Viewer also allows examining the details of a connection and can show the data for that connection through appropriate user agents commonly found in the Windows environment

such as outlook express, internet explorer etc. The dialogue between communicating hosts can also be seen in a dialogue box. User can also view the configuration file used by the packet filter.

Chapter 3

The RADIUS Protocol

PickPacket currently supports the SMTP, FTP, HTTP and Telnet protocols. It can filter packets based on network and TCP/UDP level criteria as well as application level criteria for SMTP, FTP, HTTP and Telnet protocols. The major limitation of PickPacket is that it currently does not support dynamic address allocation based networks such as dialup networks. If one is interested in monitoring activities of a particular user on networks with dynamically allocated IP addresses, the IP address allocated to the user in a session needs to be known. If the IP address allocated to such a user can not be determined then all possible IP addresses have to be monitored. This compromises the privacy of other indiviuals who are accessing and dispensing data through the network and causes waste of resources in processing uninteresting packets. Moreover, the connection of interest may not be monitored because there is an upper limit on the maximum number of sessions that PickPacket can monitor simultaneously. The capability of this tool needs to be enhanced so that it can be used in efficient manner in networks with dynamically allocated IP addresses.

In dialup network the users are given a login name by the Internet Serivice Providers(ISP). The dialup user is first authenticated before granting access to network. RADIUS is a protocol that is commonly used to authenticate dialup users. Dialup users are usually assigned IP addresses dynamically using the DHCP. Adding RADIUS support to PickPacket allows monitoring of the activities of a user whose

login name is known. This chapter briefly describes the RADIUS protocol.

3.1 RADIUS Simplified

A remote user dials a well-known phone number and the modems on both ends (user and service provider) establish a connection. The user needs to be authenticated before being granted access to the network. This is done by asking the user for a login name and a password. This is where RADIUS comes in. RADIUS is a standard communications protocol using a client/server model. The modem at provider's end are typically connected to a Network Access Server (NAS). The NAS [14] uses the RADIUS protocol to communicate over the network with a RADIUS server. The RADIUS server is asked by the NAS to authenticate the user who is dialing into one of its modem ports. The RADIUS server collects the information about the user that the NAS has forwarded to it (login name, password, asynchronous serial port number, etc.). Then the RADIUS server determines whether or not the user is allowed to connect. The result of the verification of the user's identity is sent back to the NAS where it results in either the user being connected to the NAS's serial port for further communication or being refused and the modem session terminated. RADIUS is a UDP based protocol and consists of two sub-protocols: an authentication protocol and accounting protocol. The RADIUS Authentication server listens on port 1812 and the Accounting server listens on port 1813. Figure 3.1 is a sequence diagram when a user accesses the network through the Network Access Server and later disconnects itself. The steps in this sequence are described below.

1. The Network Access Server gets the username and the password from the remote machine, encrypts this information with a shared secret key and sends this with an "Access-request" to the RADIUS Server (Authentication phase).

2. If the user and password combination is valid, the RADIUS Server sends an "Access-accept" with reply with extra information such as IP-address allocated, network mask, allowed session time etc., to the Network Access Server (Authorization phase).

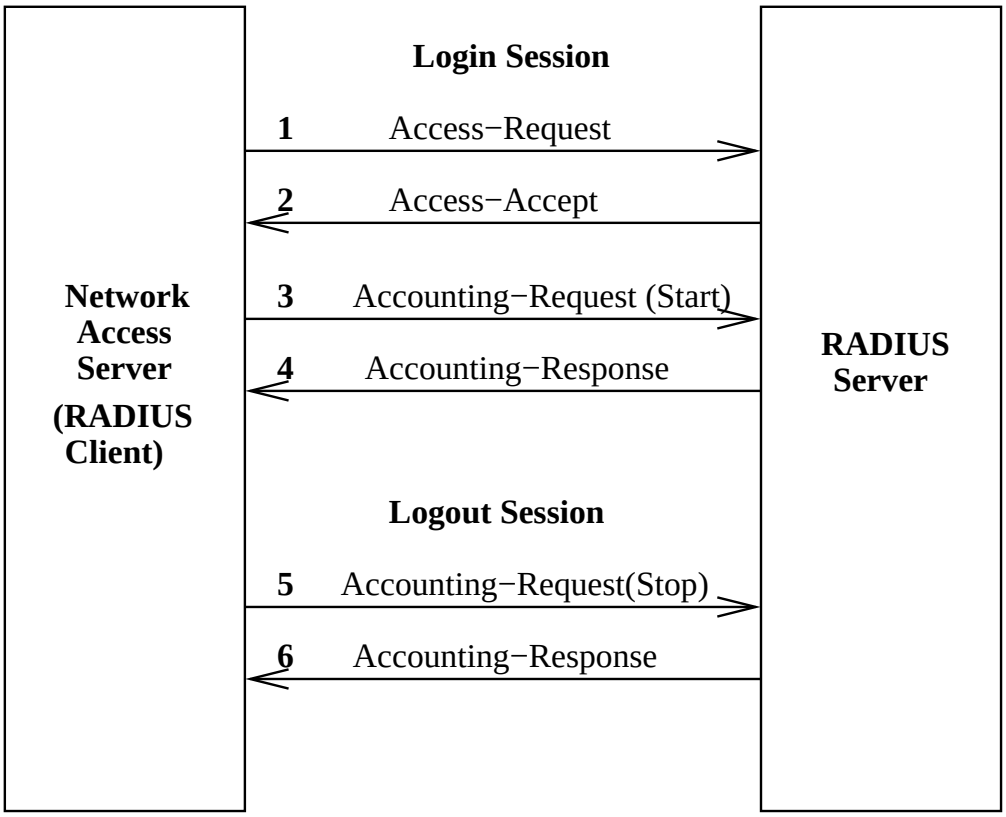


Figure 3.1: RADIUS Message Flow

3. The network Access Server sends an "Accounting-request (Start)" message to indicate that the user is logged onto the network (Accounting phase).

4. The RADIUS Server responds with an "Accounting-response" reply when the accounting information is stored.

5. When the user logs out, the Network Access Server sends an "Accounting-request (Stop)" with the information like session time, input packets, output packets etc.

6. The RADIUS Server responds with an "Accounting-response" when the accounting information has been stored.

3.1.1 RADIUS Packet Format

RADIUS packets are named requests and responses. Requests are generated by a RADIUS client and responses to requests are generated by the RADIUS server. The format of the request and response packets are same. The minimum length of RADIUS packet is 20 bytes and the maximum length is 4096 bytes. Figure 3.2 shows the RADIUS packet format.

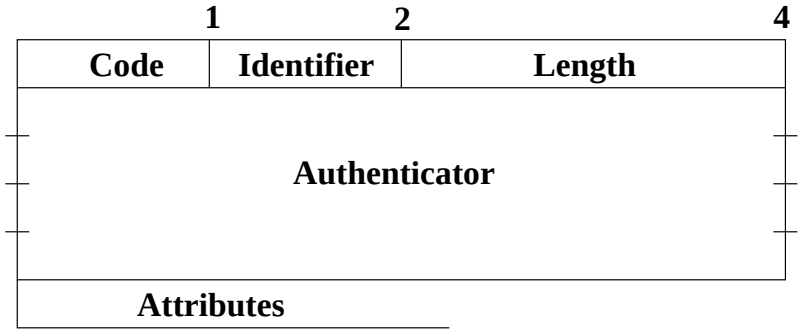


Figure 3.2: RADIUS Packet Format

The code field is one octet, and identifies the type of RADIUS packet. Some of the codes are Access-Request, Access-Accept, Access-Reject, Accounting-Request etc. The Identifier field is one octet, and aids in matching requests and replies. The length field is two octets, and contains the length of the packet including the Code, Identifier, Length, Authenticator and Attribute fields. The Authenticator field is sixteen (16) octets. The Authenticator field value is used by the RADIUS client to authenticate the reply from the RADIUS server. Attributes carry the specific authentication, authorization, information and configuration details for the request and reply. There can be zero or more attributes in a packet. Each attribute has following fields.

- Type (one octet)
- Length (one octet)
- Value (variable length)

The Type field is used to identify the type of attribute. Some of the attributes types are User-Name, Framed-IP-Address, NAS-Identifier and Session-Time etc.

The length field value gives the attribute length which includes the Type, Length and Value field. The Value field is zero or more octets and contains information specific to the attribute.

3.1.2 RADIUS Packet Types

The RADIUS Packet type is determined by the Code field in the first octet of the Packet. The packet types are Access-Request, Access-Accept, Access-Reject, Access-Challenge, Accounting-Request and Accounting-Response. These packet types are described below.

Access-Request: Access-Request packet are sent by NAS to a RADIUS server, and convey information used to determine whether a user is allowed access to NAS. The attributes which are generally sent with this type of packet are User-Name, User-Password(Encrypted), NAS-IP-Address and/or NAS-Identifier etc.

Access-Accept: Access-Accept packets are sent by the RADIUS server, and provide specific configuration information necessary to begin delivery of service to the user. The attributes which are generally sent with this type of packet are Framed-IP-Address, Reply-Message and Framed-MTU etc.

Access-Reject: Access-Reject packets are sent by the RADIUS Server. If any value of the received attributes in Access-Request is not acceptable, then the RADIUS server sends this packet. The Reply-Message attribute is a text message that is generally sent with this type of packet that the NAS may display to the user.

Access-Challenge: Access-Challenge packets are sent by the RADIUS server. The challenge collects additional data from the user. The Attributes field generally has a Reply-Message attribute. If NAS does not support challenge/response, it treats this type of response as an Access-Reject instead. If the NAS supports challenge/response, The NAS may display the text message (Reply-Message attribute content), if any, to the user, and then prompt the user for a response. It then constructs a new Access-Request which also includes the user response.

Accounting-Request: Accounting-Request packets are sent from NAS to RADIUS accounting server, and convey information used to provide accounting for a service provided to a user. The important attribute which is sent with these packets are Acct-Status-Type. It can have many values such as Accounting Start and Accounting Stop. An Accounting-Request packet with Acct-Status-Type as Accounting Start is sent at the beginning of the user service and packet with Accounting Stop is sent at the end of service. So with these two type of values of Acct-Status-Type (Start and Stop) one can know when a session starts and when it ends. The Accounting-Request packet with Acct-Status-Type attribute set to Stop also has associated attributes like Acct-Session-Time, Acct-Input-Packets, Acct-Output-Packets, Acct-Terminate-Cause etc. The Accounting-Request packet with Acct-Status-Type attribute value as Start also has associated attributes like User-Name, Framed-IP-Address, NAS-IP-Address and/or NAS-Identifier etc.

Accounting-Response: Accounting-Response packets are sent from RADIUS accounting server to NAS upon receipt of an Accounting-Request. A RADIUS Accounting-Response is not required to have any attributes in it. This is simply an acknowledgement packet that Accounting Server has successfully received and recorded the accounting packet sent by NAS. If because of any reason Accounting Server can not record the accounting packet, then it does not send this packet to NAS.

3.1.3 RADIUS Attributes

RADIUS attributes carry the specific authentication, authorization, information and configuration details for the requests and replys. The end of the list of attributes is indicated by the length of the RADIUS packet. Some of the attributes of interest to us are described below.

User-Name: This attribute indicates the name of the user to be authenticated. If available then it must be sent in Access-Request packets. This also may be present in Access-Accept and Accounting Request packet.

Framed-IP-Address: This attribute in the Access-Request makes it possible for a NAS to provide the RADIUS server with a hint of the user IP address before user authentication. The server is not required to honour the request. RADIUS Server can also tell NAS to use a specific IP address or use whatever available.

Framed-MTU: This Attribute specify the Maximum Transmission Unit to be configured for the user. This attribute can exist in Access-Request and Access-Accept packet.

Reply-Message: This Attribute indicates text which may be displayed to the user. This attribute can exist in Access-Reject, Access-Accept or Access-Challenge packet. When used in an Access-Accept, it is the success message. When used in an Access-Reject, it is the failure message. When used in an Access-Challenge, it may indicate a dialog message to prompt the user for a response.

Acct-Session-Time: This attribute records the number of seconds that the user has received service. This attribute can exist in Accounting-Request packet where the Acct-Status-Type is set to Stop.

Acct-Status-Type: This attribute specify whether Accounting-Request packet contains information about the beginning of the user service (Start) or the end (Stop) of user service.

NAS-IP-Address: This attribute gives the IP Address of the NAS which is requesting authentication of the user, and should be unique to the NAS within the scope of the RADIUS server. This attribute is used only in Access-Request and Accounting-Request packets. Either NAS-IP-Address or NAS-Identifier must be present in an Access-Request/Accounting-Request packet.

NAS-Identifier: This attribute contains a string identifying the NAS originating the Request. It is only used in Access-Request and Accounting-Request packets.

NAS-Port: This Attribute specify the physical port number of the NAS which is authenticating the user. It is only used in Access-Request packets.

Acct-Session-ID: This attribute is used to match Accounting Start and Stop packets. For a session Accounting-Request packet must have the same Acct-Session-Id. Access-Request packet also may have this attribute, if it does, then the NAS must use the same value in the Accounting-Request packets for that session.

Class: This attribute is first sent, if available, in an Access-Accept and then should be sent unmodified by the NAS to the accounting server as part of the Accounting-Request packet. This attribute is can be used offline to match Access-Request/ Accept packets with Accounting packets.

This completes the discussion on the RADIUS protocol. The next chapter covers design and implementation of RADIUS packet filter, post-processing of RADIUS packet and enhancement made to *Configuration File Generator* and *Data Viewer* component of PickPacket to support RADIUS protocol.

Chapter 4

RADIUS Support in PickPacket

This chapter discusses the modifications made to the various PickPacket components for supporting the RADIUS protocol. First the enhancements made to *Configuration File Generator* are described. Then the design and implementation of RADIUS filter and post-processor is discussed. Finally the enhancements made to the *Data Viewer* component of PickPacket are discussed briefly.

4.1 Configuration File Generator

The *Configuration File Generator* is enhanced so that the user can also specify the filtering criteria for the RADIUS protocol. These criteria are specified in a way similiar to basic criteria except that instead of source IP range the dialup user name is given. A panel is added in the GUI for specifying the RADIUS criteria in each tab used for specifying criteria for different application layer protocols. Figure 4.1 shows the new GUI screen for specifying HTTP specific criteria. The RADIUS criteria are stored in the configuration file in following format.

```
username:CaseSensitive:DestIPRange:SrcPortRange:DestPortRange:TLP:Application
```

Here userame is dialup user's login name, Application is the application layer protocol (HTTP, SMTP, TELNET, FTP etc.) which is to be monitored for the dialup user of interest and TLP is transport layer protocol (TCP/UDP). For example, if

all SMTP session of a dialup user with login name "gauravj" are to be monitored, the RADIUS criteria will look like following.

```
gauravj:N:0.0.0.0-0.0.0.0:1024-65535:25-25:TCP:SMTP
```

If any RADIUS criteria are specified then two filtering criteria are automatically added by *Configuration File Generator* to the basic criteria to capture all RADIUS authentication and accounting packets. These two criteria look like following.

```
0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1812-1812:UDP:RADIUS
0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1813-1813:UDP:RADIUS
```

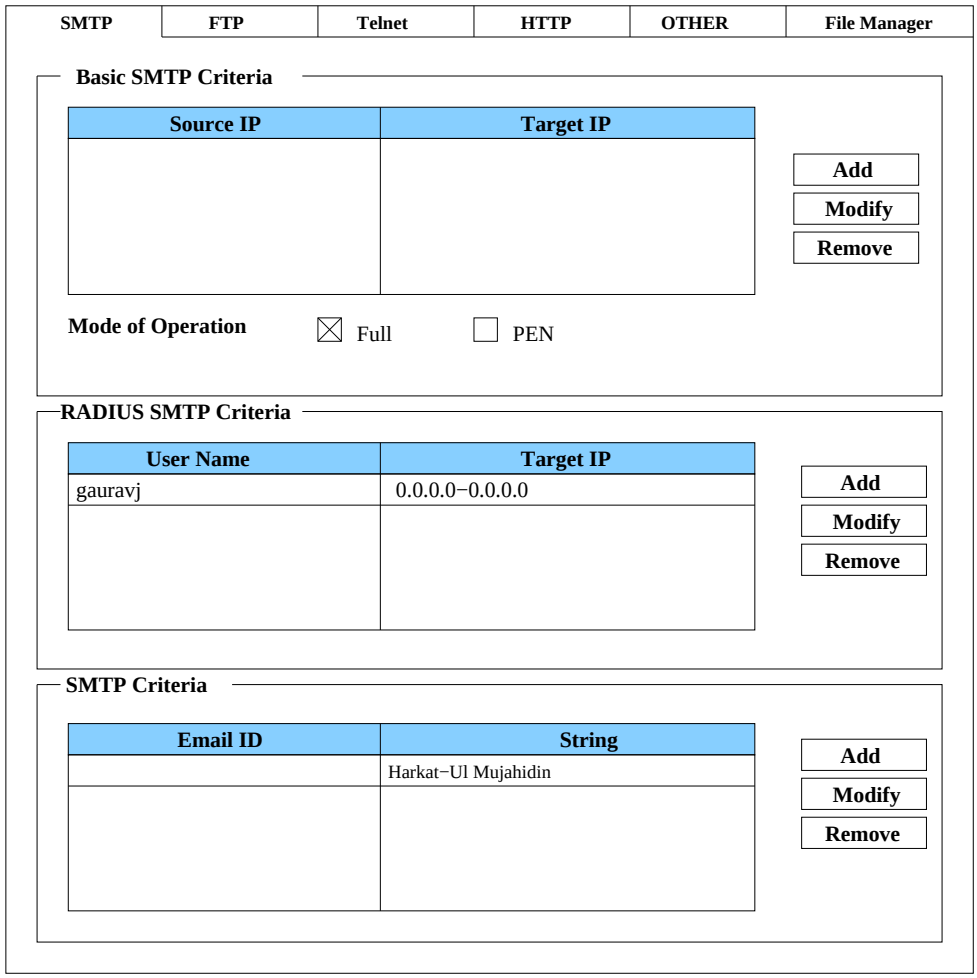


Figure 4.1: Configuration File Generator: SMTP Tab

4.2 RADIUS Filter

The RADIUS Filter captures RADIUS Packets flowing across the network. The RADIUS filter has to extract the IP address associated with the dialup users of interest. The IP address allocated to user with other details specified in the RADIUS criteria for this user are used for monitoring activites of this user. The RADIUS filter stops monitoring user activities as soon as the user of interest terminates the session.

4.2.1 Design and Implementation

The RADIUS filter maintains three hash table to keep track of RADIUS authentication and accounting requests.

1. access request hash table: This keeps only those access requests which are waiting for a reply from the RADIUS Authentication Server.

2. acct request hash table: This keeps only those accounting requests corresponding to which Accounting-Request(Stop) packets have not been seen so far.

3. temp store hash table: This keeps those access requests which have been accepted by the RADIUS Authentication server, but whose accounting has not started.

After receiving a packet the RADIUS filter first parses the packet and the attributes of the packet are stored in a structure. Further processing depends upon the type of packet. Figure 4.2 to 4.4 show the processing of some RADIUS packet types by the RADIUS Filter.

If the packet is an Access-Request or Accounting-Request(Start) packet, then the User-Name attribute is matched with usernames of interest. If a match occures then a request is added into "access request hash table" or "acct request hash table" depending upon the type of packet. If the packet has a Framed-IP-Address attribute then if any user with the same IP address is currently being monitored, then monitoring of that user is stopped as it is assumed that somehow Accounting-Request(Stop)

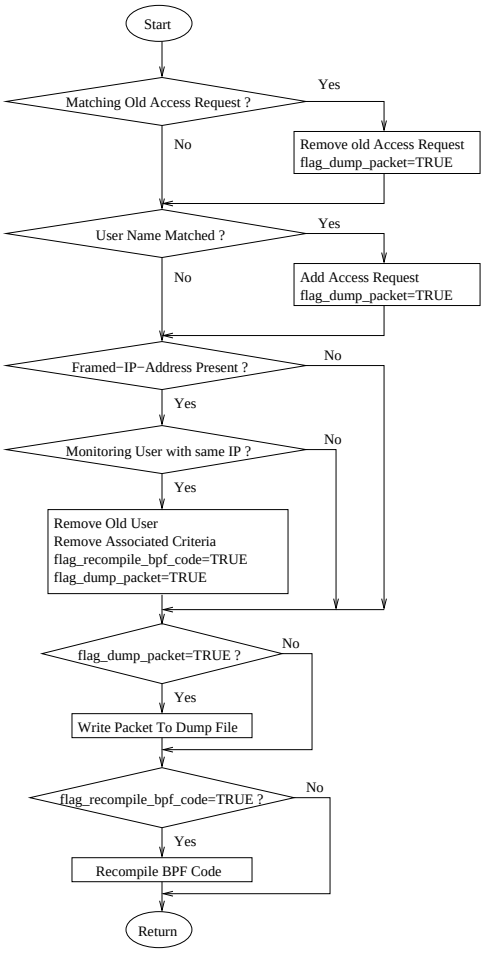


Figure 4.2: Access-Request Packet Processing by RADIUS Filter

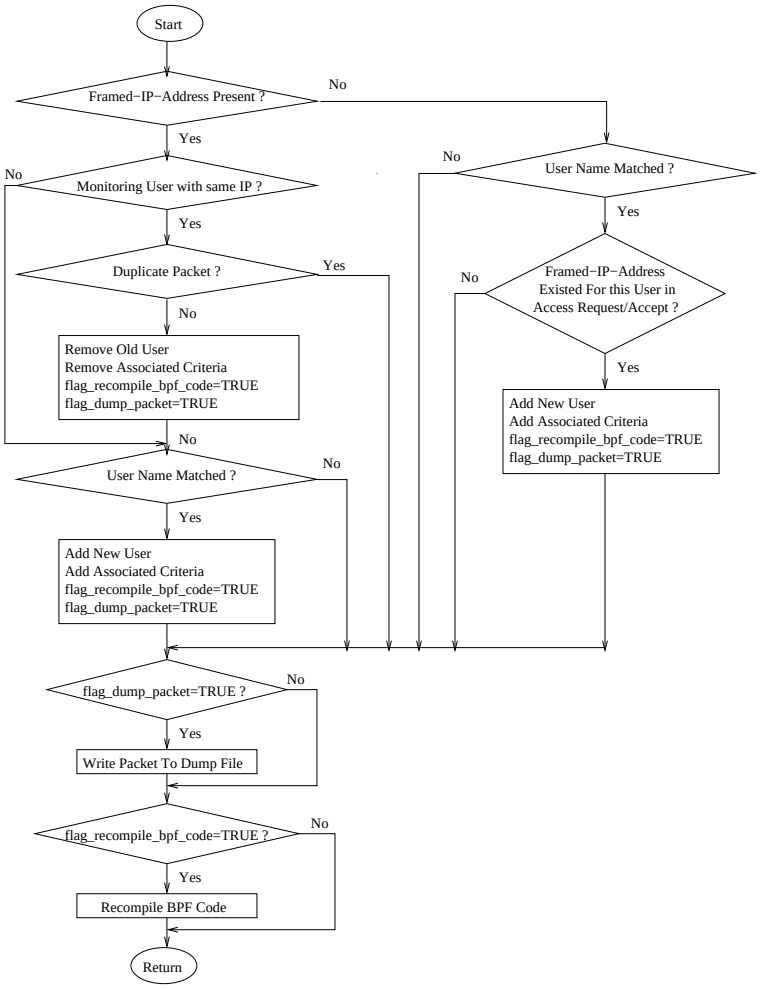


Figure 4.3: Accounting-Request(Start) Packet Processing by RADIUS Filter

packet has been missed. If the packet type is Accounting-Request(Start) then monitoring of the user is also started. If packet type is Accounting-Request(Stop) then a matching Accounting-Request(Start) is searched in the "acct request hash table". The Acct-Session-Id attribute is used for this matching. If a match is found then monitoring of this user is stopped. Accounting-Request packets with other values for A cct-Status-Type attribute are discarded. The request packets are stored in a file if username matches with any dialup users of interest, or monitoring of a dialup user starts or stops because of the request packet.

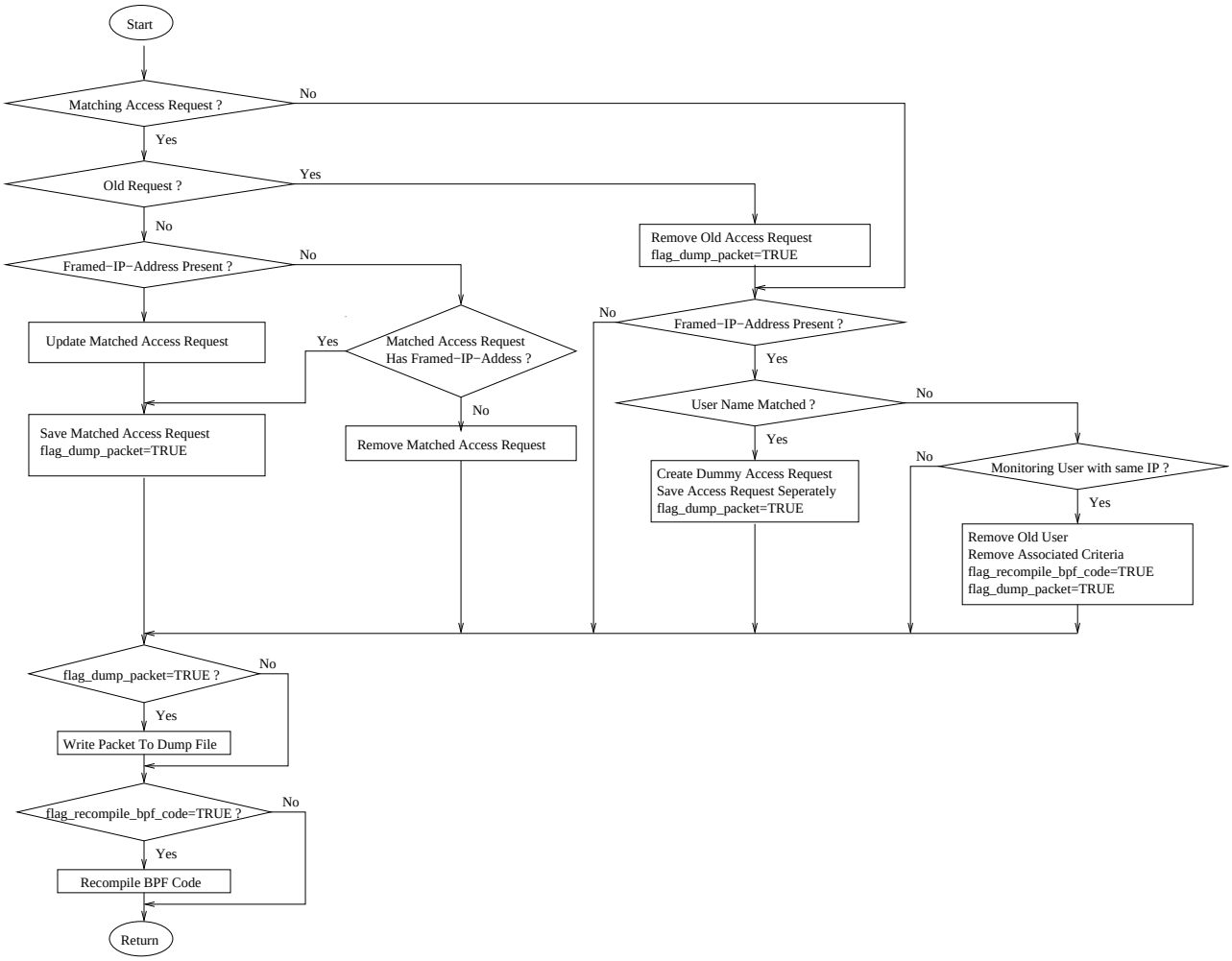


Figure 4.4: Access-Accept Packet Processing by RADIUS Filter

For a response packets the identifier field of the packet is used to find the matching request. The relevant hash tables are searched for a matching request for all response packets. If the response is either Access-Reject or Access-Accept, then access request hash table is searched. If response is Accounting-Response then, acct reqest hash table is looked into. If no matching request is found then the response packet is discarded. If a matching request is found then this packet is stored on disk as we stored the request packet earlier. If the matching request is access request then this request is also removed from access request hash table. The Access-Challenge packet is treated like Access-Reject because NAS after getting response from the

user sends another Access-Request packet to RADIUS server that finally illicits an Access-Accept response. Finally if the packet has a Framed IP Address attribute then if any user is currently being monitored with same IP address then monitoring of that user is stopped as it is assumed that somehow Accounting-Request(Stop) packet has been missed.

In all the cases monitoring of a user of interest starts by adding the user specific filtering criteria to the basic criteria and stops by removing the user specific filtering criteria from the basic criteria. The bpf code is recompiled whenever filtering criteria are added or removed to basic filter.

4.3 RADIUS Post-Processor

The RADIUS post-processor works on the output of the RADIUS filter. It needs to extract meta information that may consist of username, Framed-IP-Address, start time, end time etc. from the RADIUS packets. This information is used to find out the connections that were initiated by users authenticated using RADIUS. The post-processor also needs to detect when a RADIUS session starts and when it ends.

4.3.1 Design and Implementation

The PickPacket Post-Processor works on the output of the PickPacket filter. The *Sorter* module of the PickPacket post-processor does not need any changes for processing RADIUS packets, as this module does not have any dependency on the application layer protocols. The *Connection Breaker Module* separates out all UDP packets in a different file named on the basis of the four tuple(source IP, destination IP, source port and destination port). RADIUS packets have to be processed differently due to the following. If the RADIUS Authentication and Accounting server are running on different machines a link between various accounting and authentication packets for a session has to be established. Processing based on four tuples would store such packets on separate files. The same problem arises when RADIUS packets need to be sent to RADIUS fallback server because the main server is down or unreachable. The *Connection Breaker* module handles this problem by storing

all RADIUS packets in a single file named "radius.dump". The *Meta Information Gathering Module* first processes this file, if available, by calling the RADIUS post-processor. The RADIUS post-processor creates a RADIUS meta information file with an extension "radius".

The RADIUS post-processor reads packet one by one from "radius.dump" file and processes it. The processing depends upon type of RADIUS packet. The processing is done in a way similiar to RADIUS packet filter. The RADIUS post-processor writes meta information to RADIUS meta information file. It also creates separate conversation files for each user session of interest. These files contains both authentication and accounting packets. Figure 4.5 to 4.7 shows the post-processing of RADIUS packets.

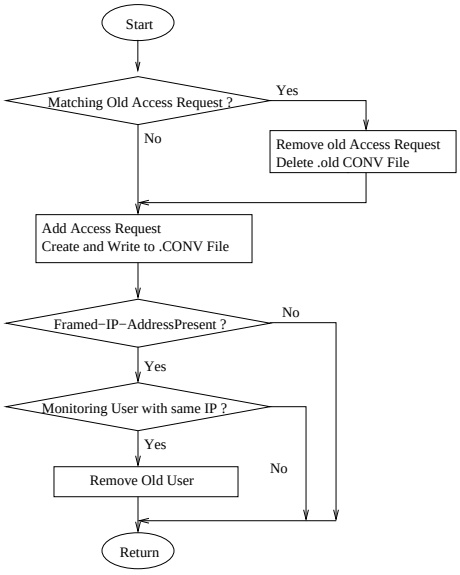


Figure 4.5: Access-Request Packet Post-Processing

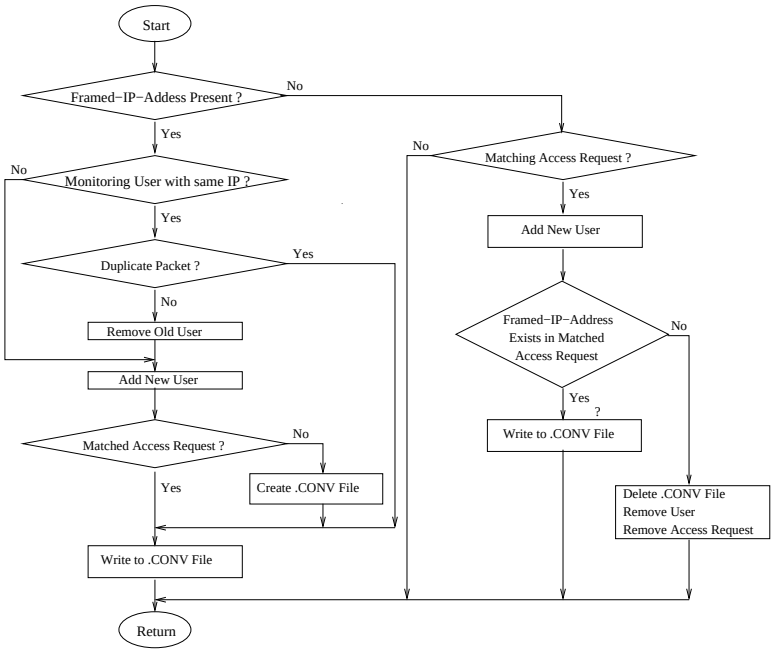


Figure 4.6: Accounting-Request(Start) Packet Post-Processing

Once all the packets have been read from the "radius.dump" file, the post processor checks whether there are some requests that have not received Accounting-Request packet with Acct-Status-Type set to Stop. It is possible that these Accounting-Request packets were missed for some reasons such as the session was on when filter

program was terminated or the dump file got exhausted etc. All such requests are now assumed to have received their corresponding Accounting-Request packets with Acct-Status-Type as "Stop".

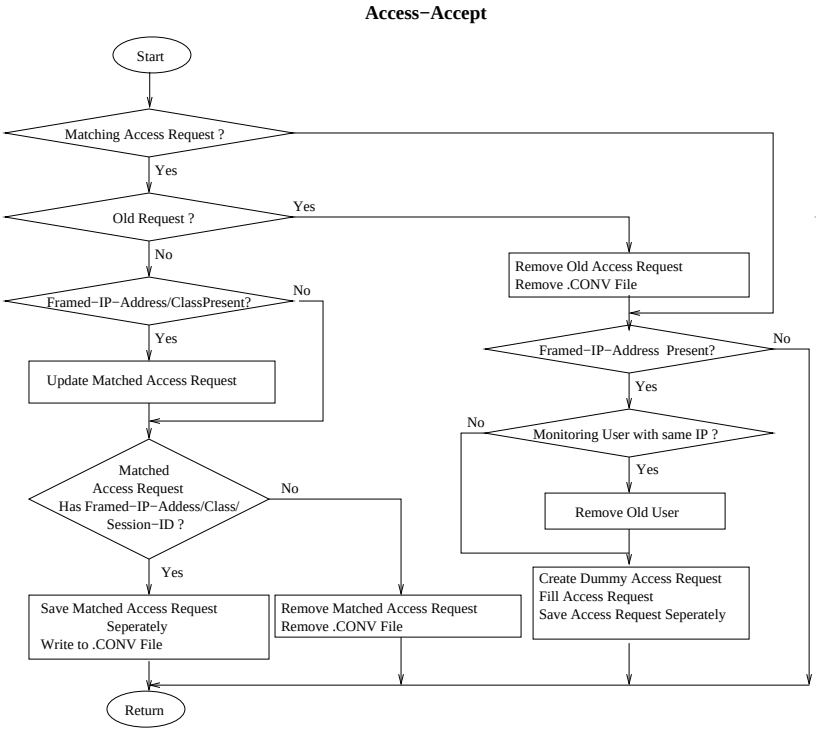


Figure 4.7: Access-Accept Packet Post-Processing

RADIUS post-processor finally generates a list of all radius sessions, which consists of allocated IP address, start and end time when this IP address was active and RADIUS connection id (a unique integer value for each RADIUS session). This list is used by all other application level protocol post processors to check whether a particular communication was initiated by a RADIUS authenticated user.

4.4 Data Viewer

The *Data Viewer* component of PickPacket is enhanced to render the RADIUS post-processed information. If a connection belongs to a RADIUS user then the RADIUS details such as start time of session, end time of session, IP address allocated to user,

session duration etc. can be viewed. Figure 4.8 shows a sample RADIUS detail form. The RADIUS dialogue box shows communication between the RADIUS client and server. It includes both authentication and accounting request and response details. All forms of the *Data Viewer* that show data for application layer protocols have have been enhanced to show RADIUS details.

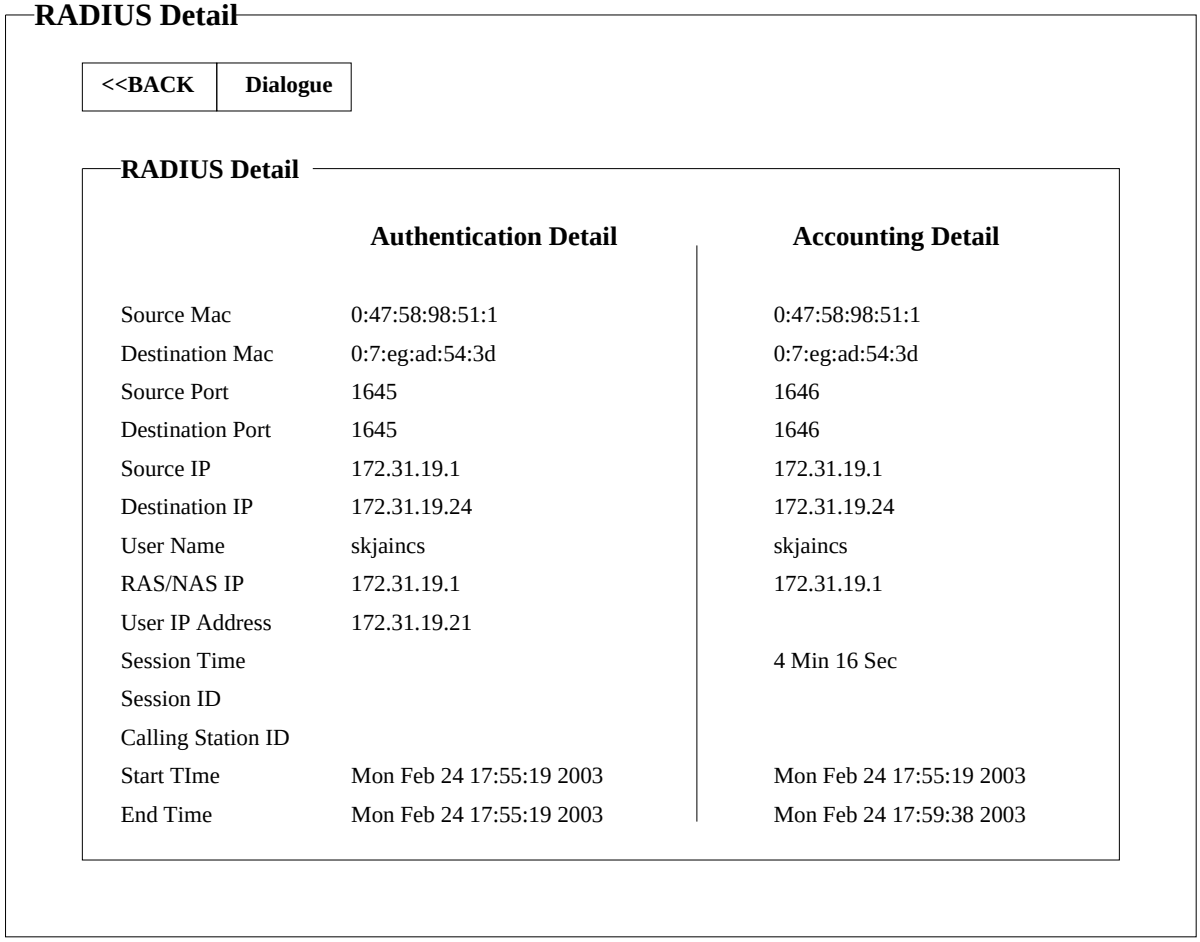


Figure 4.8: RADIUS Detail Form

Chapter 5

Testing of RADIUS Filter

In this chapter, we describe the test setup used for testing the RADIUS filter.

The initial testing of RADIUS filter program was done with the help of a utility provided by Cistron RADIUS Server [21]. This utility is called "radclient". The "radclient" is a RADIUS client program and can send arbitrary RADIUS packets to a RADIUS server, then show the received replies. Using this utility, different scenarios such as missing Accounting-Request(Stop) packet, Access-Request Packet etc. were generated to test various parts of filter program.

The final testing of RADIUS filter program was done by setting up a simple ISP environment, where a user makes a dialup connection on PPP to the ISP. After being authenticated by the RADIUS server, the user is allowed to access resources. Figure 5.1 shows the setup used for testing. One machine with Intel Pentium 2.4 GHz CPU, 256 MB RAM and running Linux kernel version 2.4.18-3 were used as RADIUS Server (both authentication and accounting server). Cistron RADIUS server [21] version 1.6 was used as RADIUS authentication and accounting server. One more machine with similar configuration was used for running the PickPacket Filter program. Another Pentium 2.4 GHz CPU machine with 256 MB RAM and running Linux kernel version 2.4.18-3 was used as the dialup user client machine. Two D-Link DMF-336/E modems were used. One modem was connected to the serial port of dialup user's machine and other was at one of WAN port of D-Link DI-540 Remote Access Server (RAS). This RAS has four WAN ports (RS-232) and

one LAN (UTP) port. A WAN port can be selected to fulfill either Internet Access, Remote Access or Lan-To-Lan Routing function. The WAN port used in setup is configured for Remote Access function. The D-Link DI-540 RAS does not send the Framed-IP-Address attribute in either Access-Request or Accounting-Request packet with Acct-Status-Type set as Start. So in this test setup RADIUS server was configured to send the IP address in Access-Accept packet that NAS is to allocate to the dialup user. The two machines running the RADIUS Server and the PickPacket Filter program respectively and the LAN port of the RAS were connected to a 10/100 Dual Speed D-Link Hub.

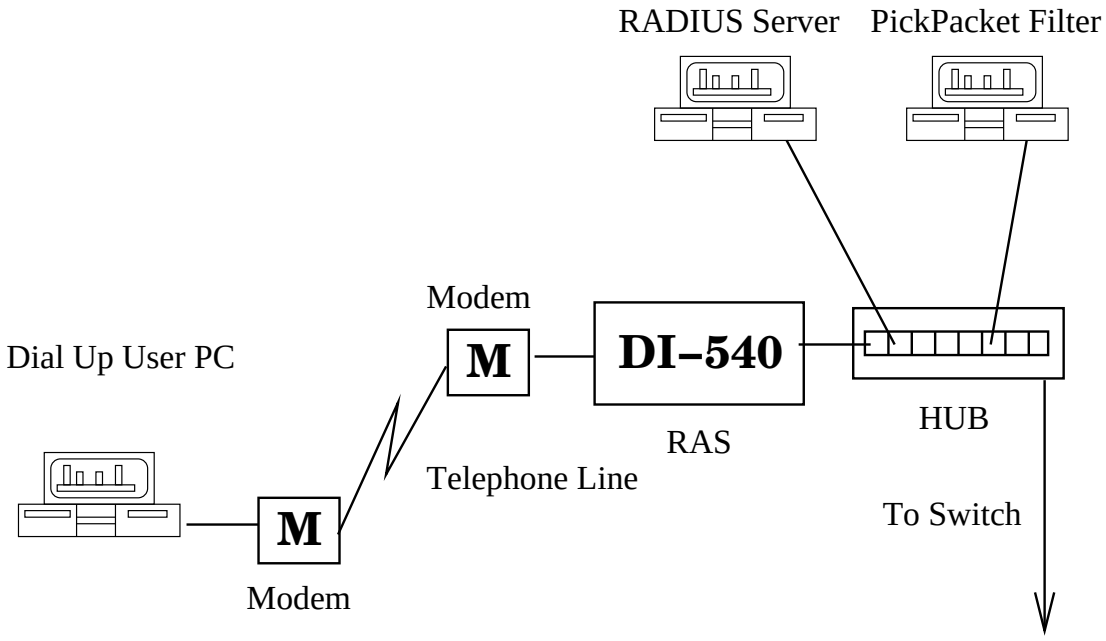


Figure 5.1: RADIUS Test Setup

The Net-Device Manager 5.83 utility [2] provided by D-Link was used to configure the Remote Access Server. The various configuration parameters used were as follows:

```
Modem Settings          - All Default
Dialup Settings         - All Default
```

```
General Settings
    WAN Port Function          - Remote Access
    Use RADIUS Authentication - Yes
    User Authentication        - PAP

Routing Table              - Routing Details Entered for CSE Router
IP Mapping                 - All Default
```

The Net-Device Monitor 5.83 utility [2] provided by D-Link was used to know the status of ports in Remote Access Server.

The kppp utility [12] was used to setup PPP connection at the dialup user's machine. The sample RADIUS Server's configuration files used are given in Appendix B. The sample configuration file for filter program is also given in Appendix B.

Chapter 6

Conclusions

PickPacket is a network monitoring tool that can capture packets flowing across the network based on a highly flexible user defined set of criteria. PickPacket allows the filtering of packets on the basis of criteria specified by the user both at the network level and the application level of the protocol stack. The design of PickPacket is modular, flexible, extensible, robust and efficient. This makes it easy to extend it to support a new application level protocol. Judicious use of PickPacket can also help protect the privacy of individuals and captures only packets which matches the criteria specified by the user onto the disk. This is not something most sniffers are capable of doing. The captured data is stored in standard tcpdump/libpcap format which offers the user a choice of using " post-processing and rendering tools" other than those provided by PickPacket.

This thesis discussed the filtering of packets based on the RADIUS application level protocol and how the information present in the RADIUS packets can be used to monitor dialup users. Users of PickPacket can now specify dialup usernames as filtering criteria for RADIUS packets. The RADIUS packets Post-capture analysis is also discussed in this thesis.

Several experiments were conducted to test the functionality of the RADIUS packet filter of PickPacket. All the bugs which appeared during the testing were fixed.

6.1 Further Work

PickPacket currently supports SMTP, FTP, HTTP, Telnet and RADIUS application level protocols. There is always scope for extending PickPacket to support other application level protocols. Currently protocol used for reading mail i.e. POP3 or IMAP are not supported. The support for these two protocols can be added. PickPacket currently support IPv4 packets. The support can be added for IPv6, the next generation Internet Protocol. The support for DHCP, which is used for assigning IP addresses for high-speed users ((cable-modems, DSL, company networks) can also be added.

PickPacket design needs some improvement so it can also take advantage of availability of multi-processor architecture on a machine. This can be the first step towards making this tool to run on Gigabit network.

References

[1] S. Prashant Aditya. "Pickpacket: Design and Implementation of the HTTP postprocessor and MIME parser-decoder". Technical report, Department of Computer Science and Engineering, IIT Kanpur, Dec 2002. http://www.cse.iitk.ac.in/research/btp2003/98316.html.

[2] "D-link Device Driver". http://www.dlink.com.au/tech/drivers/files/routers/di5xx.htm.

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

[4] R. Droms. "Dynamic Host Configuration Protocol". Technical report, 1997. http://www.ietf.org/rfc/rfc2131.txt.

[5] Gerald Combs et al. "Ethereal". Available at http://www.ethereal.com.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. "Hypertext Transfer Protocol". Technical report, 1997. http://www.ietf.org/rfc/rfc2068.txt.

[7] Robert Graham. "carnivore faq". http://www.robertgraham.com/pubs/carnivore-faq.html.

[8] "How Carnivore Works". http://www.howstuffworks.com/carnivore.htm.

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

[10] Neeraj Kapoor. "Design and Implementation of a Network Monitoring Tool". Technical report, Department of Computer Science and Engineering, IIT Kanpur, Apr 2002. http://www.cse.iitk.ac.in/research/mtech2000/Y011111.html.

[11] J. Klensin. "Simple Mail Transfer Protocol". Technical report, 2001. http://www.ietf.org/rfc/rfc2821.txt.

[12] "Linux PPP HOWTO". http://www.tldp.org/HOWTO/PPP-HOWTO.

[13] Steve McCanne 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.

[14] D. Mitton and M. Beadles. "Network Access Server Requirements Next Generation (NASREQNG) NAS Model ". Technical report, 2000. http://www.ietf.org/rfc/rfc2881.txt.

[15] Brajesh Pande. "Design and Implementation of a Network Monitoring Tool". Technical report, Department of Computer Science and Engineering, IIT Kanpur, Sep 2002. http://www.cse.iitk.ac.in/research/mtech2000/Y011104.html.

[16] J. Postel. "User Datagram Protocol". Technical report, 1980. http://www.ietf.org/rfc/rfc0768.txt.

[17] J. Postel. "Transmission Control Protocol". Technical report, Information Sciences Institute, 1981. http://www.ietf.org/rfc/rfc0793.txt.

[18] J. Postel and J. Reynolds. "Telnet Protocol Specification". Technical report, 1983. http://www.ietf.org/rfc/rfc0854.txt.

[19] J. Postel and J. K. Reynolds. "File Transfer Protocol". Technical report, 1985. http://www.ietf.org/rfc/rfc0959.txt.

[20] Boyer R. and J Moore. "A fast string searching algorithm". In *Comm. ACM 20*, pages 762–772, 1977.

[21] "Cistron RADIUS Server". ftp://ftp.radius.cistron.nl/pub/radius.

[22] C. Rigney. "Remote Authentication Dial In User Service ". Technical report, 2000. http://www.ietf.org/rfc/rfc2866.txt.

[23] C. Rigney, W. Willats, and P. Calhoun. "Remote Authentication Dial In User Service ". Technical report, 2000. http://www.ietf.org/rfc/rfc2869.txt.

[24] C. Rigney, S. Willens, A. Rubens, and W. Simpson. "Remote Authentication Dial In User Service ". Technical report, 2000. http://www.ietf.org/rfc/rfc2865.txt.

[25] Stephen P. Smith, Henry Perrit Jr., Harold Krent, Stephen Mencik, J. Allen Crider, Mengfen Shyong, 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_entry.htm.

[26] Jacobson V., Leres C., and McCanne S. *"pcap - Packet Capture Library"*, 2001. Unix man page.

Appendix A

A Sample Configuration File

```
#This is a sample configuration file
#Be very careful if you edit a configuration file manually
# The syntax should be preserved
# 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 spcifies 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 third sections specifies the number  of connections to open simultaneously
# for some applications
# The next sections describe in no particular order the application specific
# input criteria.
# This file has a fixed format Careful!!

#**************First Section*****************************
<Output_File_Manager_Settings>
```

```
   <Default_Output_File_manager_Settings>
#number of specified files
       Num_Of_Files=2
#the full file name relative/absolute will do
       File_Path=dump1.dump
#the file size in MB
       File_Size=12
       File_Path=dump2.dump
#the 0 file size means that file can be of max available size
#only the last file can have File_Size=0.
       File_Size=0
   </Default_Output_File_manager_Settings>
</Output_File_Manager_Settings>
#**************End First Section************************

#**************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 sorce IP from SrcIP1 to SrcIP2
#                   For associated ports from SrcP1 to SrcP2
# and For the range of desitnation 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, 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
# Important: Some old NAS/RAS sends packets assuming RADIUS Auth Server port
# as 1645 and Accounting Server port as 1646. So for this type of RAS/NAS we
# need to change server port
# in configuration file as mentioned in next two lines.
# Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1645-1645:UDP:RADIUS
# Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1646-1646:UDP:RADIUS
<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=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:23-23: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
   Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1024-65535:UDP:DUMP_FULL
   Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1812-1812:UDP:RADIUS
   Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1813-1813:UDP:RADIUS
</Basic_Criteria>
#**********End Second Section*****************************

#**************Third Section*****************************
# Has tunable number of connections that should be monitored
# by some applications of interest SIMULTANEOUSLY
<NUM_CONNECTIONS>
   NUM_CONNECTIONS=5
   NUM_SMTP_CONNECTIONS=500
   NUM_FTP_CONNECTIONS=500
   NUM_HTTP_CONNECTIONS=500
```

```
   NUM_TELNET_CONNECTIONS=500
   NUM_RADIUS_CONNECTIONS=500
</NUM_CONNECTIONS>
#**********End Third Section*****************************

#**************Application Specific Specifications******
#If there are RADIUS Specific criteria then those criteria comes first in this file
#**************RADIUS Specifications******
<RADIUS_Configuration>
   Num_Of_Criteria=3
   Criteria=skjaincs:no:0.0.0.0-0.0.0.0:1024-65535:1-65535:TCP:DUMP_FULL
   Criteria=vijayg:no:0.0.0.0-0.0.0.0:1024-65535:25-25:TCP:SMTP
   Criteria=vijayg:no:0.0.0.0-0.0.0.0:1024-65535:23-23:TCP:TELNET
</RADIUS_Configuration>
#**************SMTP Specifications******
<SMTP_Configuration>
   <SMTP_Criteria>
       NUM_of_Criteria=2
       <Search_Email_ID>
           Num_of_email_id=1
           Case-Sensitive=yes
           E-mail_ID=skjaincs@cse.iitk.ac.in
       </Search_Email_ID>
       <Search_Text_Strings>
           Num_of_Strings=1
           Case-Sensitive=yes
           String=book
       </Search_Text_Strings>
       <Search_Email_ID>
           Num_of_email_id=2
           Case-Sensitive=yes
```

```
            E-mail_ID=skjaincs@iitk.ac.in
            E-mail_ID=brajesh@hotmail.com
        </Search_Email_ID>
        <Search_Text_Strings>
            Num_of_Strings=0
        </Search_Text_Strings>
    </SMTP_Criteria>
    Num_of_Stored_Packets=750
    Mode_Of_Operation=full
</SMTP_Configuration>
#**********END SMTP Specifications******

#**********FTP Specifications******
<FTP_Configuration>
    <FTP_Criteria>
        NUM_of_Criteria=1
        <Usernames>
            Num_Of_Usernames=2
            Case-Sensitive=no
            Username=ankanand
            Username=nmangal
        </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>
    Num_of_Stored_Packets=750
    Monitor_FTP_Data=yes
    Mode_of_Operation=full
</FTP_Configuration>
#**********END 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>
   Num_of_Stored_Packets=750
   Mode_Of_Operation=full
</HTTP_Configuration>
#*********END HTTP Specifications******

#*********TELNET Specifications******
<TELNET_Configuration>
   <Usernames>
       Num_of_Usernames=1
       Case-Sensitive=yes
       Username=ankanand
   </Usernames>
   Mode_Of_Operation=full
</TELNET_Configuration>
#*****END TELNET Specifications******

#*********TEXT SEARCH Specifications******
#These have to be added manually

<TEXT_Configuration>
   <Search_Text_Strings>
       Num_of_Strings=1
       Case-Sensitive=no
       String=timesofindia
   </Search_Text_Strings>
   Mode_Of_Operation=pen
</TEXT_Configuration>
#*****END TEXT SEARCH Specifications******

#**********End Application Specific Specifications****
```

Appendix B

Configuration Files used for RADIUS Filter Testing

B.1 Files for testing RADIUS filter

B.1.1 PickPacket Filter Configuration File

```
#DI-540 D-Link Remote Access Server assumes that RADIUS server are listening on
#port 1645 and 1646.
<Output_File_Manager_Settings>
    <Default_Output_File_manager_Settings>
    Num_Of_Files=1
    File_Path=/dev/null
    File_Size=4000
    </Default_Output_File_manager_Settings>
</Output_File_Manager_Settings>
<BASIC_CRITERIA>
    DEVICE=eth0
    Num_Of_Criteria=2
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1645-1645:UDP:RADIUS
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1646-1646:UDP:RADIUS
</BASIC_CRITERIA>
```

```
<NUM_CONNECTIONS>
   NUM_CONNECTIONS=5
   NUM_SMTP_CONNECTIONS=50
   NUM_FTP_CONNECTIONS=50
   NUM_HTTP_CONNECTIONS=50
   NUM_TELNET_CONNECTIONS=50
   NUM_RADIUS_CONNECTIONS=50
</NUM_CONNECTIONS>
<RADIUS_Configuration>
  Num_Of_Criteria=10
  Criteria=skjaincs:no:0.0.0.0-0.0.0.0:1024-65535:25-25:TCP:SMTP
  Criteria=skjaincs:no:0.0.0.0-0.0.0.0:1024-65535:20-20:TCP:FTP
  Criteria=skjaincs:no:0.0.0.0-0.0.0.0:1024-65535:21-21:TCP:FTP
  Criteria=murthyj:no:0.0.0.0-0.0.0.0:1024-65535:3128-3128:TCP:HTTP
  Criteria=kanth:no:0.0.0.0-0.0.0.0:1024-65535:25-25:TCP:SMTP
  Criteria=goyals:no:0.0.0.0-0.0.0.0:1024-65535:20-20:TCP:FTP
  Criteria=goyals:no:0.0.0.0-0.0.0.0:1024-65535:21-21:TCP:FTP
  Criteria=murthyj:no:0.0.0.0-0.0.0.0:1024-65535:3128-3128:TCP:HTTP
  Criteria=brajesh:no:0.0.0.0-0.0.0.0:1024-65535:25-25:TCP:SMTP
  Criteria=neeraj:no:0.0.0.0-0.0.0.0:1024-65535:25-25:TCP:SMTP
</RADIUS_Configuration>
<SMTP_Configuration>
   <SMTP_Criteria>
       NUM_of_Criteria=1
       <Search_Email_ID>
           Num_of_email_id=1
           Case-Sensitive=yes
           E-mail_ID=skjaincs
       </Search_Email_ID>
       <Search_Text_Strings>
           Num_of_Strings=2
```

```
            Case-Sensitive=no
            String=Harkat-Ul Mujahidin
            String=Biological Terrorism
        </Search_Text_Strings>
    </SMTP_Criteria>
    Num_of_Stored_Packets=100
    Mode_Of_Operation=full
</SMTP_Configuration>
<FTP_Configuration>
    <FTP_Criteria>
        NUM_of_Criteria=1
        <Usernames>
            Num_Of_Usernames=1
            Case-Sensitive=no
            Username=skjaincs
        </Usernames>
        <Filenames>
            Num_Of_Filenames=0
        </Filenames>
        <Search_Text_Strings>
            Num_Of_Strings=1
            Case-Sensitive=no
            String=Procedure Call
        </Search_Text_Strings>
    </FTP_Criteria>
    Num_of_Stored_Packets=100
    Monitor_FTP_Data=yes
    Mode_of_Operation=full
</FTP_Configuration>
<HTTP_Configuration>
    <HTTP_Criteria>
```

```
        NUM_of_Criteria=0
    </HTTP_Criteria>
    <Port_List>
        Num_of_Ports=2
        HTTP_Server_Port=80
        HTTP_Server_Port=3128
    </Port_List>
    Num_of_Stored_Packets=0
    Mode_Of_Operation=full
</HTTP_Configuration>
```

B.1.2 Cistron RADIUS Server Configuration Files

```
File Name - clients
   Client Name       Key
   172.31.19.1       dialup (DI-540 RAS)
   172.31.19.25      testing123 (Machine used with radclient utility)
File Name - naslist
   NAS Name          Short Name    Type
   172.31.19.1       nas01         other (DI-540 RAS)
   172.31.19.25      nas02         other (Machine used with radclient utility)
File Name - users
   For user - skjaincs
       skjaincs  Auth-Type = Local, Password = "ksgajm123"
       Service-Type = Framed-User,
       Framed-Protocol = PPP,
       Framed-IP-Address = 172.31.19.7,
       Framed-IP-Netmask = 255.255.252.0,
       Framed-Routing = Broadcast-Listen,
       Framed-Filter-Id = "std.ppp",
       Framed-MTU = 1500,
       Framed-Compression = Van-Jacobson-TCP-IP
```

```
For user - kanth
    kanth  Auth-Type = Local, Password = "kanth123"
    Service-Type = Framed-User,
    Framed-Protocol = PPP,
    Framed-IP-Address = 172.31.19.21,
    Framed-IP-Netmask = 255.255.252.0,
    Framed-Routing = Broadcast-Listen,
    Framed-Filter-Id = "std.ppp",
    Framed-MTU = 1500,
    Framed-Compression = Van-Jacobson-TCP-IP
For user - brajesh
    brajesh  Auth-Type = Local, Password = "brajesh123"
    Service-Type = Framed-User,
    Framed-Protocol = PPP,
    Framed-IP-Address = 172.31.19.22,
    Framed-IP-Netmask = 255.255.252.0,
    Framed-Routing = Broadcast-Listen,
    Framed-Filter-Id = "std.ppp",
    Framed-MTU = 1500,
    Framed-Compression = Van-Jacobson-TCP-IP
For user - neeraj
    neeraj  Auth-Type = Local, Password = "neeraj123"
    Service-Type = Framed-User,
    Framed-Protocol = PPP,
    Framed-IP-Address = 172.31.19.23,
    Framed-IP-Netmask = 255.255.252.0,
    Framed-Routing = Broadcast-Listen,
    Framed-Filter-Id = "std.ppp",
    Framed-MTU = 1500,
    Framed-Compression = Van-Jacobson-TCP-IP
For user - murthyj
```

```
    murthyj  Auth-Type = Local, Password = "murthyj123"
    Service-Type = Framed-User,
    Framed-Protocol = PPP,
    Framed-IP-Address = 172.31.19.24,
    Framed-IP-Netmask = 255.255.252.0,
    Framed-Routing = Broadcast-Listen,
    Framed-Filter-Id = "std.ppp",
    Framed-MTU = 1500,
    Framed-Compression = Van-Jacobson-TCP-IP
For user - goyals
    goyals  Auth-Type = Local, Password = "goyals123"
    Service-Type = Framed-User,
    Framed-Protocol = PPP,
    Framed-IP-Address = 172.31.19.3,
    Framed-IP-Netmask = 255.255.252.0,
    Framed-Routing = Broadcast-Listen,
    Framed-Filter-Id = "std.ppp",
    Framed-MTU = 1500,
    Framed-Compression = Van-Jacobson-TCP-IP
```

Appendix C

Structure of various Data Viewer Input Files

Here we describe the structure of the various application protocols' connection record files. We follow this description by a sample packetfile(someid.pkt).

C.1 Structure of Connection Record Files

1. someid.smtp (The SMTP connection record file)
2. someid.http_cr (The HTTP connection record file)
3. someid.http_dr (The HTTP connection detail record file)
4. someid.other (Non-HTTP/SMTP/Telnet/FTP connections record file)
5. someid.ftp_pdr (The FTP data connection record file)
6. someid.ftp_pcr (The FTP control connection record file)
7. someid.telnet (The Telnet control connection record file)
8. someid.radius (The RADIUS record file)
9. someid.signature (The Signature file)

These files may or may not be present depending on what was sniffed For instance, the someid.http_cr and someid.http_dr file may not be there if no HTTP connections were sniffed for.

The details of fields are as follows:

(A) someid.smtp

1. Connection ID
2. Source Mac Address
3. Destination Mac Address
4. Source IP
5. Destination IP
6. Source port
7. Destination Port
8. Conversation file name (*.CONV)
9. Server dialogue file name (*.S)
10. Client dialogue file name (*.C)
11. Eml file name
12. Date at which the mail was sent (in "date" command format)
13. Emailid of sender
14. Number of persons to whom the mail was sent
15. Comma separated list of mailids of all the recipients of this mail
16. Subject of mail
17. Start time (seconds from Unix epoch) of the connection
18. Start time (micro seconds from Unix epoch) of the connection
19. End time (seconds) of the connection
20. End time (micro seconds) of the connection
21. RADIUS Connection ID

(B) someid.http_cr

1. Connection ID
2. Source Mac Address
3. Destination Mac Address
4. Source IP
5. Destination IP

6. Source port
7. Destination Port
8. Conversation file name (*.CONV)
9. Server dialogue file name (*.S)
10. Client dialogue file name (*.C)
11. hostname
12. Start time (seconds) of the connection
13. End time (seconds) of the connection
14. RADIUS Connection ID

(B) someid.http_dr

1. Connection ID
2. Source Mac Address
3. Destination Mac Address
4. Source IP
5. Destination IP
6. Source port
7. Destination Port
8. Conversation file name (*.CONV)
9. hostname
10. Request Method
11. URI
12. Request Data file name (POST Data)
13. Dummy Field
14. Response Data file name
15. Start time (seconds) of the connection
16. End time (seconds) of the connection
17. Post Data Encoding (Urlencoding, multipart/form-data)
18. Boundary (POST data encoded in multipart/form-data)

(C) someid.other

1. Connection ID
2. Source Mac Address
3. Destination Mac Address
4. Source IP
5. Destination IP
6. Source port
7. Destination Port
8. Transport Protocol (udp/tcp)
9. Conversation file name (*.CONV)
10. Server dialogue file name (*.S)
11. Client dialogue file name (*.C)
12. Date at which the session was started (in "date" command format)
13. Start time (seconds) of the connection
14. Start time (micro seconds) of the connection
15. End time (seconds) of the connection
16. End time (micro seconds) of the connection
17. RADIUS Connection ID

(D) someid.ftp_pdr

1. Connection ID
2. Source Mac Address
3. Destination Mac Address
4. Source IP
5. Destination IP
6. Source port
7. Destination Port
8. Conversation file name (*.CONV)
9. FTP username
10. FTP password
11. Actual file name of file transferred
12. New file name

13. Type of the file as determined from the file command
14. Start time (seconds) of the connection
15. Start time (micro seconds) of the connection
16. End time (seconds) of the connection
17. End time (micro seconds) of the connection
18. Structure of the file transfer
19. Type of file transfer
20. Mode of file transfer

(E) someid.ftp_pcr

1. Connection ID
2. Source Mac Address
3. Destination Mac Address
4. Source IP
5. Destination IP
6. Source port
7. Destination Port
8. Conversation file name (*.CONV)
9. Server dialogue file name (*.S)
10. Client dialogue file name (*.C)
11. Date at which the session was started
12. Date at which the session ended
13. RADIUS Connection ID

(F) someid.telnet

1. Connection ID
2. Source Mac Address
3. Destination Mac Address
4. Source IP
5. Destination IP
6. Source port

7. Destination Port
8. Transport Protocol (udp/tcp)
9. Conversation file name (*.CONV)
10. Server dialogue file name (*.S)
11. Client dialogue file name (*.C)
12. Date at which the session was started (in "date" command format)
13. Start time (seconds) of the connection
14. Start time (micro seconds) of the connection
15. End time (seconds) of the connection
16. End time (micro seconds) of the connection
17. RADIUS Connection ID

(G) someid.radius

1. Connection ID
2. Source Mac Address
3. Destination Mac Address
4. Source IP
5. Destination IP
6. Source port
7. Destination Port
8. Conversation file name (*.CONV)
9. Dummy Field
10. Dummy Field
11. User Name
12. NAS/RAS IP Address
13. NAS/RAS Port
14. Framed IP Address (Dynamically allocated IP to user)
15. Acct Session Time
16. Session ID
17. Class
18. Calling Station ID

19. Start time (seconds) of the connection
20. Start time (micro seconds) of the connection
21. End time (seconds) of the connection
22. End time (micro seconds) of the connection
23. Record Type ("A" - Accounting, "T"- Authentication)

(H) someid.signature

1. Dump File Name
2. Dump File Size
3. Signature Creation Time
4. Signature (Hex Format)

In each of the above files, the various records end with a CRLF. The individual fields are separated by a ; (semicolon).

Sample packetfile

100.cfg
101.signature
102.smtp
105.ftp_pcr
106.ftp_pdr
115.telnet
120.radius