Reconstruction of Web-Based Email in PickPacket

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

by

Vinaya Natarajan

to the

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

May, 2006

Certificate

This is to certify that the work contained in the thesis entitled "*Reconstruction of Web-Based Email in PickPacket*", by *Vinaya Natarajan*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May, 2006

[illegible]

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

Abstract

The Internet is fast becoming one of the most popular means of communication because it is fast, cheap and accessible. Its low cost and pervasiveness also make it attractive for use by criminals and terrorists. Thus there is a need to monitor network traffic. However, this monitoring should not be at the cost of the privacy of individuals. PickPaket is a flexible network monitoring tool that achieves these conflicting aims. It is a passive monitoring tool that sniffs online packets on the network and captures packets that match the criteria specified by the user. It then analyzes the captured packets offline based on the application level protocol that the packet payload belongs to. It reconstructs the original connections from the captured data and displays them in a user-friendly format. While PickPacket has a built-in support for several application level protocols like HTTP, FTP, SMTP, POP, IMAP, Telnet, Yahoo chat and IRC, it can also filter and capture traffic belonging to unsupported protocols.

While PickPacket supports capture and analysis of HTTP traffic, one important class of HTTP traffic needs further attention. Web-based email, which is basically email transfer using HTTP, is one of the most popular forms of email communication today. While PickPacket can be configured to capture all web-based email data, displaying this captured data to the PickPacket user in its original form is a challenging problem. This is because not all data gets transferred as simple HTTP pages that can be displayed as is. Also, PickPacket captures a lot of web-based email data, most of which is not of interest to the user. Some mechanism for automatic classification of the captured data is needed. This thesis investigates the problems involved in reconstruction of web-based email traffic and describes the solution implemented.

Acknowledgments

I take this opportunity to express my gratitude to my thesis supervisor Dr. Dheeraj Sanghi for his guidance, encouragement and support throughout my thesis work. I would also like to thank Dr. Deepak Gupta who, despite being on a sabbatical, was ready to guide us whenever necessary. And of course my thesis partner, Devendar Bureddy, for the year long supply of ideas, support, help and fun. I would also like to thank everybody at Prabhu Goel Research center - Chinmay, Dungara, Satyam and Palak, for making it such a cool place to work in. Also Sudheer and Ananth, my seniors, for the gentle introduction to PickPacket and their patient explanation of its architecture and functioning.

This work was carried out as part of a project that is financially supported by the Ministry of Communications and Information Technology, Government of India. Their support is duly acknowledged.

I would also like to express my gratitude and admiration for the faculty of CSE department for showing me what passion for work means and for inspiring, motivating, and improving me. I would, a little unfairly perhaps, like to collectively thank the entire MTech 2004 batch. Thank you for being so much fun, for supporting and encouraging my crazy ideas, for making me do things I would have never dared to do elsewhere!

Finally, I would like to thank my family for always being there. Their support, help and encouragement is the reason I am here today.

Contents

List of Figures

Chapter 1

Introduction

With the Internet becoming ubiquitous, it is fast becoming one of the most popular mediums of communication all over the world. With Internet services becoming accessible to the common man in our country, it has become an easy and attractive medium of communication for criminals as well. Lack of tools to monitor Internet traffic further encourages its use for illegal activities. Thus there is a need to develop and deploy tools that can monitor Internet traffic and detect such misuse. Such tools are also useful for companies who wish to monitor their network traffic for anti-company activities. However, one necessary and somewhat contradictory requirement of any such tool is that it should not compromise the privacy of individuals.

Monitoring tools work by passively listening for packets on the network and filtering them based on user-specified rules. Tools which provide the facility of specifying simple rules for filtering packets are called *packet filters.* Tools that filter packets based on complex rules and analyze the captured data are called *network monitoring tools.* Such tools understand network applications and can thus support application level filtering rules.

1.1 Network Monitoring Tools

Network monitoring tools are used to monitor data flowing across the network. They are passive tools that sniff packets on the network and capture packets that match some rules specified by the user. Network monitoring tools are usually application-aware, that is they can analyze application protocol data. Thus, packets can be filtered based on application level rules.

The network monitoring tool runs on a host machine and listens to the traffic on the network. By default, the network interface card of a host machine passes on only those packets to the operating system which are destined for itself. Since we want to monitor all traffic, the network card of the host is run in "promiscuous mode". In this mode, the network interface copies all packets that arrive on the network. Hubs were used in earlier LANs. Since hubs copy all traffic to all ports, a monitoring tools could listen to all the network traffic by simply connecting to a port of the hub. But now switches have almost completely replaced hubs in LANs. Switches forward packets only to the port to which the destination host is connected. Thus for network monitoring, special switches that support "port mirroring" are needed. These switches have a mirror port on which traffic received by other ports of the switch can be copied. The host that runs the network monitoring tool can then connect to this port and listen to network traffic.

The power and flexibility of network monitoring tools come from the ability to specify filtering rules. Filtering rules can be specified at two levels. First level filtering is based on TCP/IP criteria like IP addresses and port numbers. This level of filtering can be performed at the kernel level and is quite efficient. For example, BPF (Berkley Packet Filter) [14] is an in-kernel filter that filters packets based on a directed acyclic Control Flow Graph method. The second level of filtering rules can specify application level criteria like HTTP host names, SMTP email addresses, search strings etc. Packets that match these rules are stored on disk. This captured data is further decoded and presented in human readable form.

Several sniffers are available in commercial and public domains. *Tcpdump* [11] is a UNIX based command line sniffer. It supports on-line filtering criteria based on IP addresses and port numbers. *Windump* [6] is a version of Tcpdump for windows.

Ethereal [7] also sniffs packets from the network based on filtering criteria. However, it has better packet decoding capabilities. *Carnivore* [8, 9, 17], a packet capturing system, has been developed by FBI. It is designed to collect information about the electronic communication to or from a specific user targeted in the investigation. Carnivore is quite different from other tools as it performs filtering of packet based on a wide range of filtering criteria specific to application level protocols. It functions through wire-taps across gateways and ISPs. It is capable of monitoring users who work on dynamic IP address based networks. It has capabilities to search the application-level content for specific strings. However, most of the tools described above are designed for network management related tasks. They do not support a powerful rule specification language.

1.2 PickPacket

PickPacket is a Network Monitoring Tool being developed at Indian Institute of Technology Kanpur for the last four years. PickPacket listens to the network traffic and selects some packets for storage and further analysis. The selection/filtering criteria for the packets can be specified at all three levels - IP addresses and ports numbers, application specific parameters like email-ids, user names, URLs and search strings for the application payload.

PickPacket supports the following application layer protocols - HTTP, FTP, Telnet, SMTP, POP, IMAP, RADIUS, IRC, Yahoo chat and instant messages. The user can specify application specific criteria as well as search strings for each application protocol separately. Also, there are two levels of granularity at which the PickPacket captures packets. In "FULL" mode the whole connection data is stored while in "PEN" mode a minimal amount of information about the connection is stored. Judicious use of these features can help protect the privacy of individuals. The captured data is analyzed off line during post processing. The data is separated into TCP sessions. The application session is then reconstructed and displayed to the user. The user can view the reconstructed application data as well as the raw connection data.

1.3 Reconstructing Web-Based Email Sessions

Email is one of the most heavily used forms of communication on the Internet. While specialized protocols like SMTP, POP and IMAP are defined for mail exchange, web-based email services like Yahoo and Rediffmail account for a considerable amount of the email traffic in our country. Hence capture and display of web-based email is of immense interest to PickPacket users. While this might seem trivial given that PickPacket can be configured to capture HTTP traffic, there are several challenges in reconstructing the mail session. The aim is to show to the PickPacket user, minus unnecessary images, the exact screen that the original user must have seen. This thesis describes these challenges and presents a general solution for reconstruction of web-based email sessions.

The main challenge relates to the display of hidden data, i.e. data that does not get transferred as an HTML page that can be displayed as is by the browser. We need mechanisms to correlate this kind of data with the source of the HTML page that generates it. We also need a mechanism to populate the HTML page with the data. Secondly, web-based email services follow no standards, every service provider has developed his own way of doing things. Thus any solution to the reconstruction problem has to be general enough to accommodate the reality that new email service providers come up everyday and existing ones may change the way they do things anytime.

While PickPacket provides a powerful mechanism to filter out data that the user is not interested in, the user still ends up with a considerable amount of data to be analyzed. Since majority of the web traffic is HTTP, some mechanism for further filtering of HTTP data would be very useful. This is possible in case of web-based emails since we are working with pages from a known domain. This thesis also presents a mechanism for automatic filtering of web-based email data.

1.4 Organization of the Report

This report describes the extension of PickPacket for reconstruction of web-based email sessions. Chapter 2 discusses the architecture and design of PickPacket. Chapter 3 discusses the problems in reconstruction of web-based email traffic. Chapter 4 describes the solution and implementation. Chapter 5 discusses the testing results.

Chapter 2

PickPacket: Architecture and Design

This chapter briefly discusses the architecture and design of PickPacket. Design and implementation issues are discussed in detail in References [1, 12, 13, 16].

2.1 Architecture

PickPacket consists of four components which together capture, analyze and display data of interest to the user.

- *PickPacket Configuration File Generator* is a Java based GUI that is used to accept filtering criteria for PickPacket from the user. The user can specify IP addresses, port numbers, RADIUS criteria, application specific criteria and keywords separately for each application protocol. The criteria are written to configuration files in a format expected by the PickPacket Filter. This component can run on Windows or Linux.

- *PickPacket Filter* is the online filtering component. It receives packets from the Ethernet card of the machine it runs on and filters them based on criteria specified in the configuration files generated by the Configuration File Generator. The filtering happens at multiple levels and packets which pass through all levels are saved to disk. The Filter component runs on Linux.

- *PickPacket Post-Processor* performs offline analysis of packets stored by the Filter. It basically reconstructs TCP connections from the captured data. Data belonging to each connection is stored in a separate directory along with corresponding meta data. The PostProcessor component runs on Linux.

- *PickPacket Data Viewer* provides a web based GUI for viewing post processed data. It provides the user access to the reconstructed data, connection meta data as well as raw packet data. The DataViewer component runs on a PHP enabled web server and can be accessed from any machine that has a browser.

These components can be installed on four separate machines. It is also possible for some of the components to reside on the same machine. The PickPacket Filter, which is the online component, should typically reside on a separate machine. The PickPacket PostProcessor and the DataViewer can be installed on the same machine so that PostProcessor's output does not have to be transferred across the network.

PickPacket architecture is shown in Figure 2.1. The figure shows each component of PickPacket and the interaction between them.

2.2 Design

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

2.2.1 PickPacket Configuration File Generator

PickPacket Configuration File Generator is a Java based GUI component. It is used by the user to accept filtering criteria from the user. The user can specify TCP/IP as well as application specific criteria for each protocol supported by PickPacket. PickPacket also supports limited filtering of packets belonging to unsupported protocols. The GUI has separate tabs for each protocol where the following criteria can be entered:

- *IP address*: A specific address or an address range can be supplied.

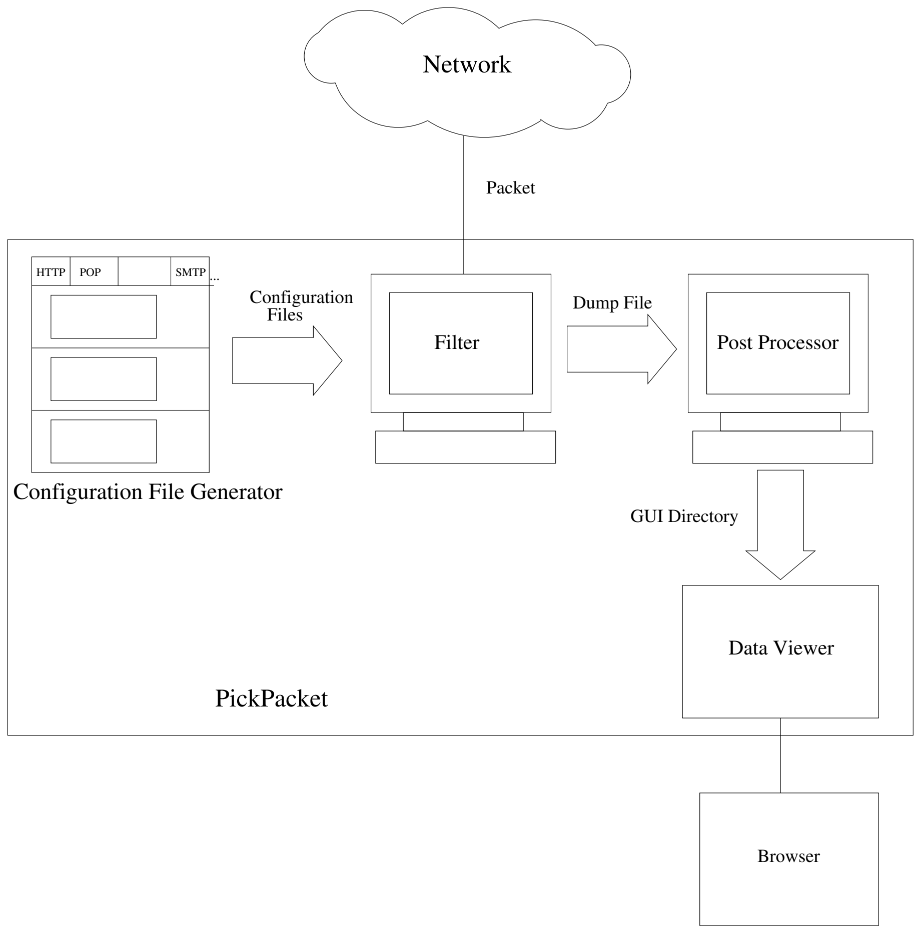


Figure 2.1: Architecture of PickPacket

- *Port Numbers*: Port numbers other than the default port number for the application protocol can be specified.
- *RADIUS criteria*: Radius user names for which traffic is to be monitored can be supplied.
- *Application specific criteria*: Host names, email addresses, chat IDs etc can be specified.
- *Keywords*: Strings to be searched for in the application data can be specified.

PickPacket supports combining of filtering criteria using AND and OR operators thus giving users a lot of flexibility in specifying what they are interested in.

The Configuration File Generator writes the criteria into two files, one which contains filter configuration parameters and the other which contains filtering rules. The first file has an extension of *.bcfg* and contains the maximum number of simultaneous connections PickPacket should monitor for each application protocol. It also contains the maximum number of packets PickPacket will keep in memory for each connection to check for filtering criteria matches. These parameters should be carefully set based on the available system memory and the expected amount of traffic for the application protocol so that no packets are dropped from memory and no connections are missed.

The filtering rules are saved in a file with extension *.pcfg*. This file has three sections:

1. The first section deals with the output files into which PickPacket Filter will write the captured packets. The user can specify the prefix to be used for the output file. The output file rolls over periodically so that captured data can be post processed. The roll over criteria can either be the file creation time or maximum file size and is specified in this section.

2. The second sections lists the IP addresses and TCP ports that the user is interested in monitoring along with the application protocol.

3. The third section contains application specific criteria. It has one subsection for each application level protocol. The subsection contains application specific criteria like hostnames, email addresses and keywords to be searched for in the application data. Whether the Filter should operate in PEN mode or FULL mode for that protocol is also specified.

2.2.2 The PickPacket Filter

The Filter is the online sniffing component. It listens to the network traffic and stores packets that match the filtering criteria. The Filter supports two transport layer protocols - UDP and TCP. Since UDP is a connectionless protocol UDP packets are treated independent of each other. All UDP packets are stored to disk. In case of TCP, which is a connection oriented protocol, the Filter keeps track of TCP connections. It sequences packets that belong to a connection, keeping them in memory to facilitate a search for keywords across packet boundaries. If a packet matches any of the filtering criteria, all packets belonging to that connection are saved to disk.

▮ *Filtering levels*

Filtering of packets is done at the following levels:

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

In kernel filters [14] are used for basic filtering. This makes filtering based on network parameters very efficient because only those packets that match the filtering criteria get copied from kernel space to user space. The next two levels of filtering are done in user space by PickPacket's application layer filters. Each application level protocol has its own filter that understands the application's data format. Thus, a new application protocol can be added to PickPacket by simply defining a filter for it.

▮ *Filter modules*

The PickPacket Filter consists of the following modules:

- *Filter generator* reads the basic filtering criteria from PickPacket's configuration files and generates corresponding BPF code. This code is used to perform in kernel filtering.

- *Basic Filter* reads packets from the network and performs in-kernel filtering based on the code generated by the Filter Generator. Packets which do not match the criteria are dropped.

- *Demultiplexer* receives packets that satisfy the basic criteria from the basic Filter. Its function is to deliver the packet to the appropriate Connection Manager based on the application level protocol to which the packet belongs. For this it makes use of the filtering rules specified in the configuration files. It uses the four tuple {Source IP, Destination IP, Source Port, Destination Port} to decide which Connection Manager the packet should be forwarded to.

- *Connection Manager* is associated with every application level protocol supported by PickPacket. It maintains TCP connection information. It keeps in memory, packets belonging to connections that might be of interest. It performs sequencing on these packets based on their TCP sequence numbers, taking care of fragmented, overlapping and out of order TCP packets. Once a packet matches some filtering criteria, the Connection Manager sets a flag for the connection that causes all packets belonging to that connection to be stored on disk.

- *Application Level Filter* receives packets from the Connection Manager. It checks the packet to see if it matches any of the application level criteria. If it does, the Application Level Filter alerts the Connection Manager which then stores all past packets belonging to that connection that are in memory. The Connection Manager will also store all future packets that belong to the connection. In case of certain protocols like FTP, dynamic filtering criteria need to be attached to the Basic Filter on the fly. The Application Level Filter for such protocols passes corresponding filtering criteria to the Filter Generator which then generates BPF code and attaches it to the Basic Filter.

- *Output File Manager* gets packets that are to be stored on disk. It writes packets to the output file, taking care to roll over to the next output file based on the criteria specified in the configuration file.

Figure 2.2 shows how data flows between the Filter modules described above.

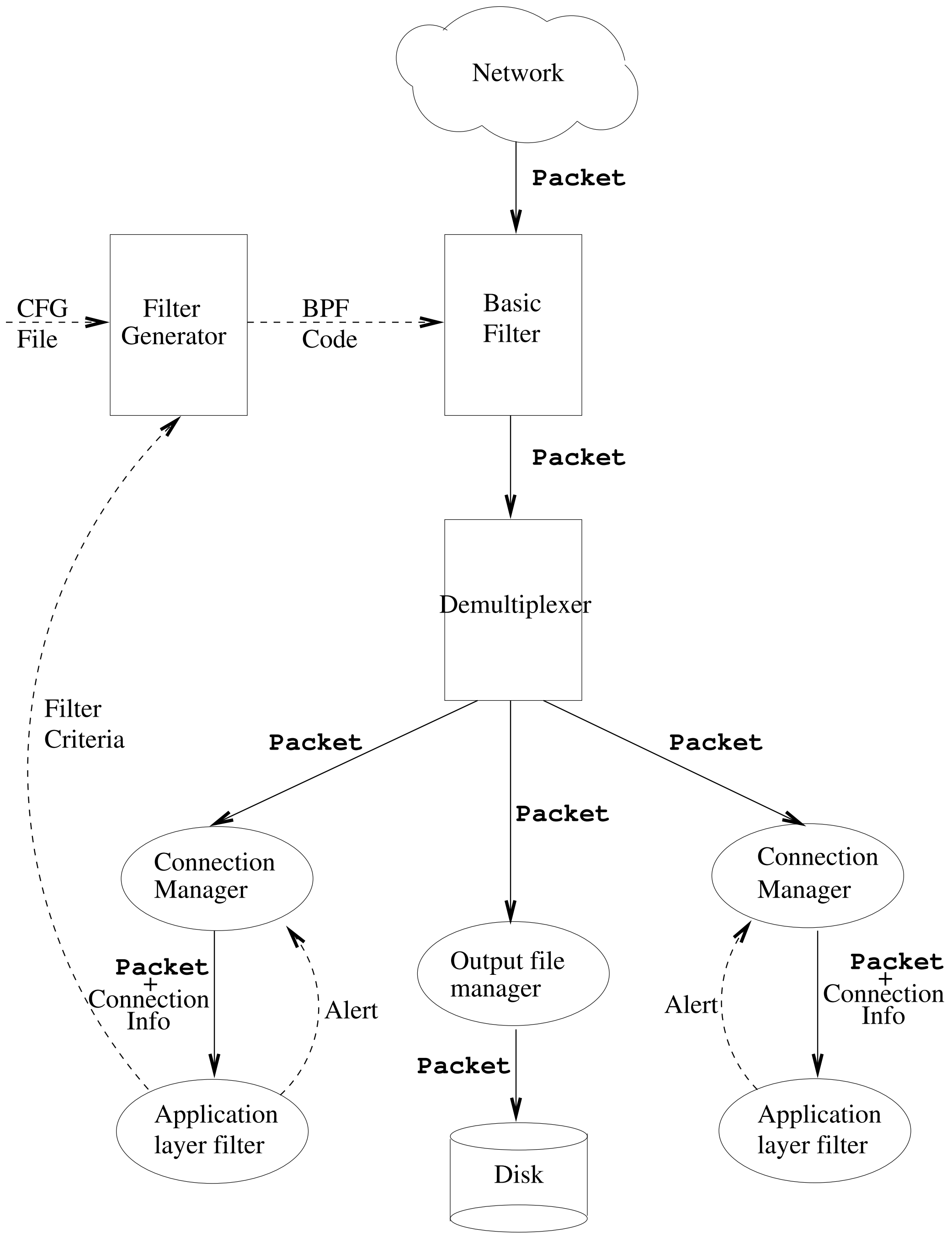
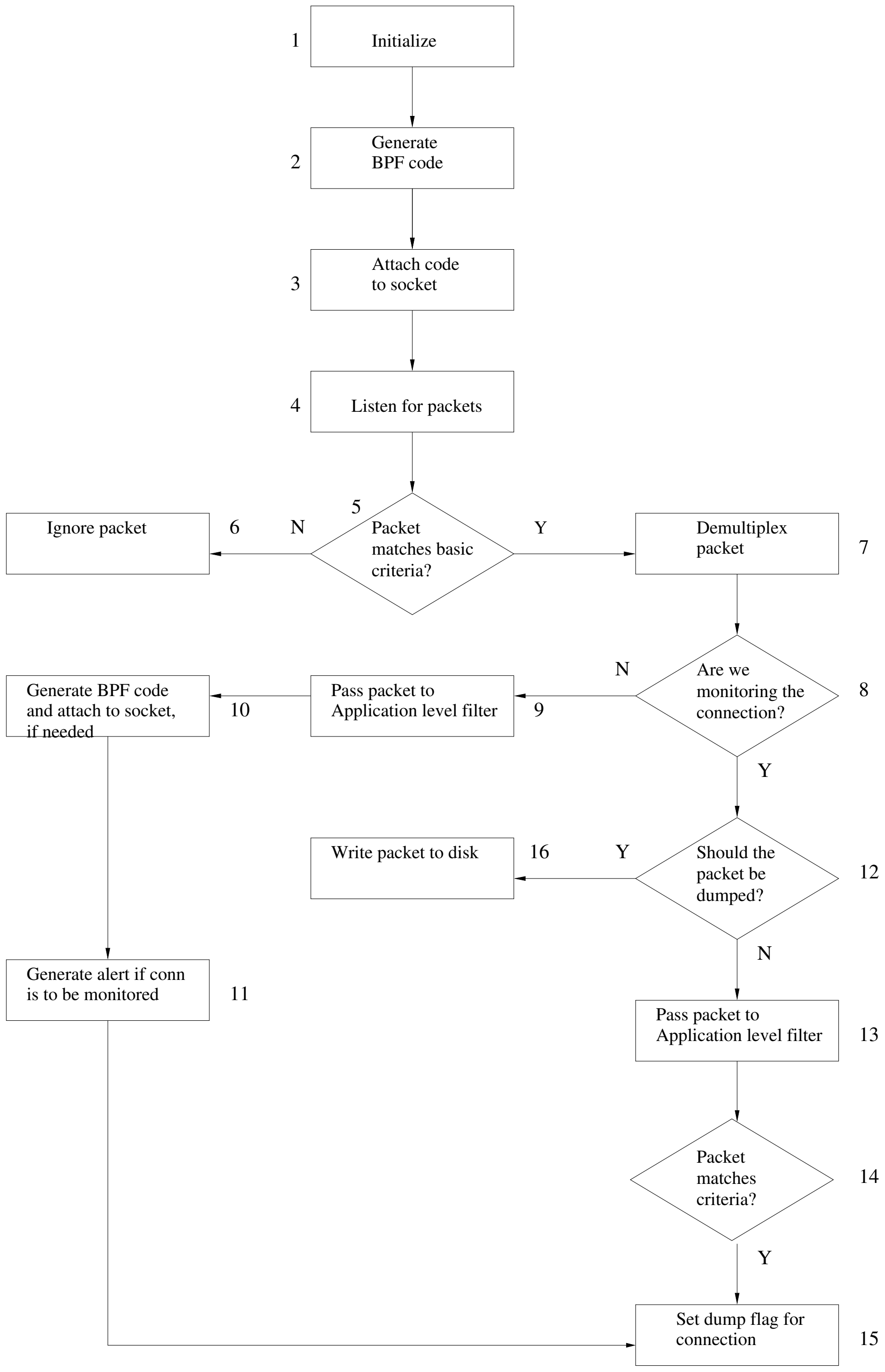


Figure 2.2: Data Flow in Pickpacket

▮ *Control Flow*

We now describe the control flow within PickPacket Filter.

1. The PickPacket configuration files are read in and the remaining modules of the Filter are initialized.
2. The BPF code for the filtering criteria specified in the configuration file is generated.
3. The filtering code is attached to the socket listening on the network.
4. The socket listens to traffic on the network in promiscuous mode.
5. When a packet is received from the network, it is passed on to the kernel. Here, the BPF code executes and checks whether the packet satisfies the filtering criteria.
6. Packets that do not match the filtering criteria are ignored (not processed any further)
7. Packets that match the filtering criteria are passed on to the Demultiplexer. The Demultiplexer decides which application protocol filter should handle the packet based on the criteria specified in the configuration file. The packet is then passed on to the Connection Manager for the appropriate protocol.
8. The Connection Manager determines whether it is monitoring the TCP connection to which the packet belongs.
9. If the connection is not being monitored (which will be the case when the first packet for a connection is received), the packet is passed on to the corresponding application level filter.
10. In certain application level protocols like FTP, filtering criteria need to be attached to the Basic Filter on the fly. In such cases, the BPF code for the criteria is generated and attached to the Basic Filter.



1
Initialize
2
Generate BPF code
3
Attach code to socket
4
Listen for packets
5
Packet matches basic criteria?
N
Y
6
Ignore packet
Demultiplex packet
7
Are we monitoring the connection?
8
N
Y
Pass packet to Application level filter
9
Generate BPF code and attach to socket, if needed
10
Should the packet be dumped?
12
Y
N
Write packet to disk
16
Generate alert if conn is to be monitored
11
Pass packet to Application level filter
13
Packet matches criteria?
14
Y
Set dump flag for connection
15

Figure 2.3: Control Flow in Pickpacket

11. The application filter looks at the packet and determines if connection is of interest. If so it *alerts* the Connection Manager to monitor all future packets belonging to the connection.

12. If the connection is being monitored, the Connection Manager checks whether the dump flag for the connection is set. The dump flag indicates that packets belonging to the connection should be written to disk. It is set when the the packet matches any of the filtering criteria.

13. If the dump flag is not set, the Connection Manager passes the packet to the corresponding application level filter.

14. The application filter checks whether the packet matches any filtering criteria. This includes application specific criteria like host names, email addresses and a search for keywords in the packet payload. The filter takes care of the case where the keyword of interest crosses packet boundary.

15. If the packet matches any filtering criteria, the application filter sets the dump flag for the connection so that the Connection Manager can store all future packets belonging to this connection.

16. If the dump flag for a connection is set, the Connection Manager writes the packet to the output file.

As the filter is an online component, it has to process packets at least at the speed at which then they arrive on the wire. Keyword search is the most computationally expensive job of the Filter. PickPacket provides a text string search library that uses the *Boyer-Moore* [5] string matching algorithm. The library provides efficient functions for case sensitive and case insensitive search.

2.2.3 PickPacket Post-Processor

PickPacket PostProcessor processes the output file generated by the Filter and converts the captured data into a format that is required by the Data Viewer and can

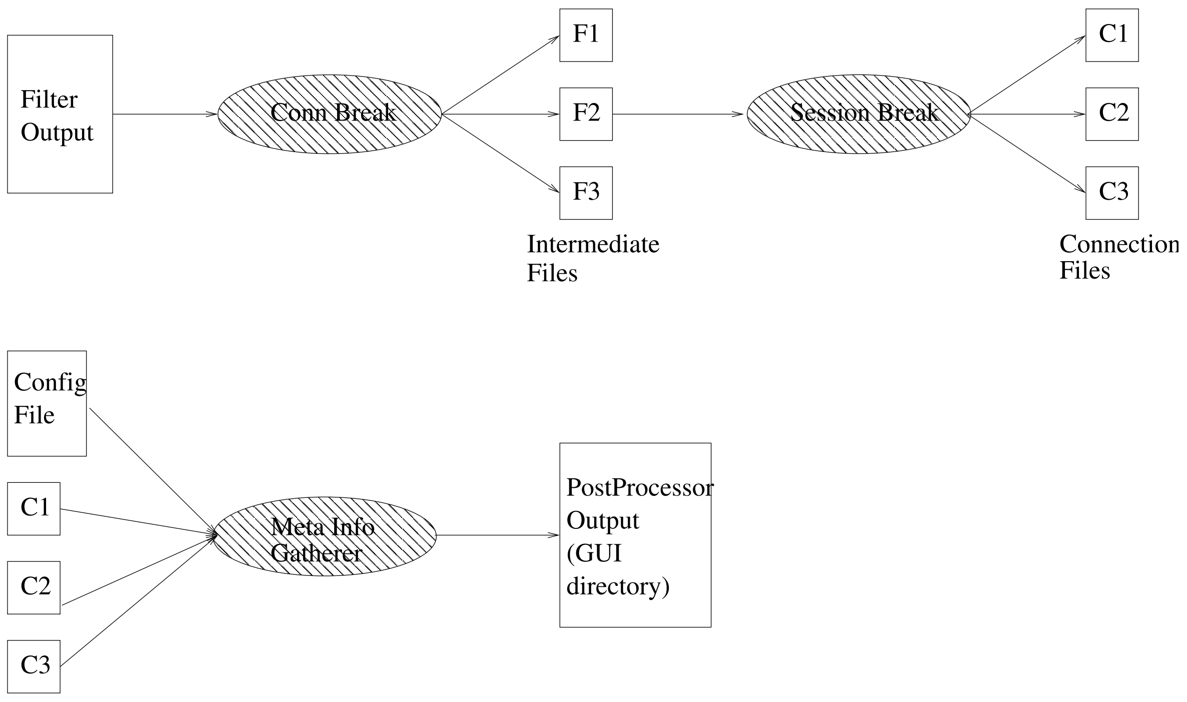


Figure 2.4: Post-Processing Design

be understood by the user. It also extracts network, transport and application layer related meta information from the connection.

The PostProcessor consists of three modules as shown in Figure 2.4.

- *Connection Breaker* separates the packets in the output file into connections. Each connection is defined by a four tuple - source IP, source port, destination IP and destination port. The Connection Breaker creates one file for each four tuple into which it writes all packets that match the four tuple.

- *Session Breaker* takes each file generated by the Connection Breaker and separates it into multiple files if it contains packets belonging to multiple sessions.

- *Meta Information Gathering Module* collects network, transport and application layer information about the connection. It creates a separate directory for each connection and writes the raw connection data, the reconstructed data and the meta information to it. Meta data extraction being an application protocol specific task, this module contains a separate sub-module for each application layer protocol supported in PickPacket.

The output of the PostProcessor is a directory whose name ends with the string "‘_gui"’ and hence is referred to as the GUI directory. It contains information about all the connections present in the output file generated by the Filter with each connection having its own sub-directory.

2.2.4 PickPacket Data Viewer

PickPacket DataViewer is used to display post-processed information to the user. The DataViewer reads the GUI directories created by the PostProcessor and displays the connection information present in them along with meta data. The DataViewer is written in PHP and runs on a web server. We use the Apache web server, but any web server that supports PHP should work. The functioning of the DataViewer can be divided into three steps:

1. *Dump Selection*: In this step, the DataViewer displays a list of GUI directories present in its "*Data*" directory. The path to the *Data* directory is configured during install.

2. *Connection Selection*: Once the user selects a GUI directory, the DataViewer reads and displays information about the connections present in the GUI directory. A GUI directory can typically contain thousands of connections. The DataViewer can be configured to display a fixed number of connections per page, say N. The DataViewer reads only N connections at a time from the disk. It also maintains a cache of recently viewed connections in memory, each entry of the cache being a set of N connections.

3. *Display of Connection Details*: When a user selects a connection, the DataViewer displays connection details. This includes network and transport layer details, application specific details, meta data and link to raw packet data.

Apart from this basic functionality, the DataViewer also provides several features that make it easier for the user to scan though captured data.

- The connections in a GUI directory can be sorted on various fields like Application Protocol name, capture time, keywords present in the application payload etc.

- An output file typically contains thousands of connections and it becomes difficult for the user to go through all of them. DataViewer provides the facility of further filtering these connections. This filtering can be based on network parameters and application protocol criteria. Once the user has applied a filter, only those connections that match the filtering criteria are displayed. The user is also given the option of temporarily hiding certain connections.

Chapter 3

Reconstruction of Web-Based Email: The Problem

3.1 Introduction

Web-based email is basically the use of HTTP to transfer emails. Despite the presence of specialized protocols for mail transfer like POP, IMAP and SMTP, web-based email remains one of the most popular forms of email communication. This is because they provide ease of access, allowing the user to check mail from anywhere without requiring an email client. Some of the most popular web-based email service providers in our country are Yahoo, Hotmail, Rediffmail and recently, Gmail. Each of these providers have their own web interface for email transfer.

Since web-based email is HTTP traffic, PickPacket can be easily configured to capture the email pages. Once the pages have been captured by the Filter, it is the job of the PostProcessor to reconstruct the email session. By reconstruction, we mean that the PickPacket user should be able to see, minus some images, what the original user must have seen on his screen from the time he logged in to the time he logged out.

In this chapter, we present the problems in reconstructing web-based email traffic. Before we get into the problems, we define certain terms that are used in describing the problem.

- *Client* is the browser end of the HTTP communication between two hosts.
- *Server* is the HTTP server that listens on Port 80 for requests from clients and sends back stored or generated HTML pages in response. In our case, it mostly refers to the web server of the web-based email service provider. For example, www.yahoo.com is the server for Yahoo, www.rediffmail.com for Rediffmail.
- *HTTP Request* is sent from the client to the server and typically contains the URL of the HTML file that the client is interested in. It may also contain data that the client wants to send to the server.
- *HTTP Response* is sent from the server to the client and typically contains the HTML file requested by the client.
- *POST* is one of the methods defined for submitting data using an HTTP Request. It is generally used when the request contains data that is likely to change the state of the server database. For example, when a user composes and sends an email, the email contents are transmitted to the server as POST data.
- *PickPacket user* is the person who uses PickPacket to capture and analyze network traffic. He thus sets up PickPacket configuration parameters, runs the Filter, PostProcesses the filter output and goes though the resulting data using the Data Viewer.
- *(Web-based) Email user* is the person using a web-based email service to send and receive mails. The PickPacket user is interested in capturing the HTTP traffic generated by the email user.

3.2 Hidden Data

One of the main challenges in reconstructing web-based email pages is reconstruction of pages that accept some text input from the user. Examples of such pages are:

- The login page, where the username and password are accepted.

- The compose page where email addresses, subject and body of the mail are required as input.

Consider the common case of a web-based email user composing a mail. When the email user types in the address, subject, content, etc., and submits the page, this data is sent to the server using some standard method defined by HTTP (generally POST). Though this communication between the client and the server is captured by PickPacket, it is captured as a request sent by the client to the server. And while the Dataviewer gives the user the option of viewing HTTP requests, viewing email contents this way is not convenient. The problem is more severe in case the email user sends attachments, which are also sent as HTTP POST requests. Word documents, PDF files, images, etc., cannot be displayed in the browser as part of an HTTP request. Such files can only be displayed by corresponding viewers when the attachment content is extracted from the HTTP request into a file of its own.

As far as email addresses, subject and the text component of an email is concerned, the most natural way of displaying this data is to display the "‘compose"’ page with the corresponding fields filled up using POST data from the HTTP Request. There are two issues to be solved here:

1. Finding out the compose page that generated the HTTP Request containing email data.

2. Correlating the POST data in the HTTP Request with the fields of the compose page.

3.3 Developing a General Solution

Web-based email services follow no standards. Each email service provider has his own way of doing things. And the providers are free to change their interface anytime. Also, new web-based mail service providers are coming up everyday. Hence any solution for reconstruction of web-based emails has to be general enough to accommodate new players as well as tolerant of changes made by existing ones. Hence it has to based on the basic features that are most likely to be common across all email service providers. For example, a solution that assumes that a certain URL is used for certain task is not general, easily extensible or resistant to change.

Also, each service provider has his own idiosyncrasies that need to be considered. For example, consider an email user who is composing a mail and now wants to add an attachment. In case of Yahoo, when the user clicks on "Add Attachment", a draft of the composed mail is saved on the server. This save draft request is sent to the server as an HTTP Request, with the data entered in the compose page so far forming POST data of the request. When we try to correlate HTTP requests that contain such POST data with the corresponding compose page, we should be careful not to use data from this draft request. This is because the actual mail contents might be different from the draft contents.

3.4 Handling AJAX Based Services

AJAX or Asynchronous JavsScript And XML is a web development technique that is fast being adopted by web-based email services providers. It is a combination of several existing technologies like XML, HTML, CSS and JavaScript. In traditional web applications, every user action generally results in an HTTP request from the client to the server. The server does the processing required by the request and sends data back to the client. Every request thus results in a page being loaded into the browser. AJAX does things a little differently [3]. When a web page is loaded, an AJAX engine is also loaded along with it, usually in some hidden frame. The AJAX engine is JavaScript code that is used to communicate with the server and to display the data sent by the server in the browser. The advantages of this technique are:

1. The requests made by the engine to the server as a result of some email user action are asynchronous, that is, the user does not have to wait for the response to come back. He can continue to view the rest of the page and can also perform actions that generate further (asynchronous) requests.

2. Since the display logic is now on the client side, the server does not have to

send an entire HTML page in response to each request. It simply sends the "data" for the page and the AJAX engine is responsible for rendering this data. For example, if the email user has clicked on an email link, the server needs to send back only the contents of the mail. The AJAX engine running on the browser is responsible for displaying the email in a suitable format.

Reconstruction of pages belonging to mail services that uses AJAX has to be handled differently from that of traditional email services. This is because in traditional email services the entire HTML page is captured and can be displayed as is by the browser. In case of AJAX however, what is captured is the "data" for the page. This data is not in any standard format. Every mail service uses a different format for the data and uses its own engine to interpret and display it. We need to develop a general solution for display of AJAX data.

3.5 Uninteresting Pages

Consider that a PickPacket user wants PickPacket to monitor an email address that has been provided by some web-based email service. PickPacket should ideally capture all mails sent to and from that account. To achieve this, the PickPacket user cannot simply supply the email address as one of the HTTP keywords and capture pages that contain the address. This is because it is not necessary that the user's email address be contained in every communication (TCP connection) between client and server. Thus, PickPacket has to be configured to captures all packets that are exchanged between the client and the web-based email server. This results in a lot of unnecessary pages being captured. It would make the PickPacket user's job a lot easier if he can be given some indication about which of the captured pages might be of interest to him.

3.6 HTTP Requests From Captured Pages

PickPacket DataViewer provides the PickPacket user the option of viewing the captured HTTP page. However, the captured page might have code that generates

HTTP requests for images, scripts, stylesheets and even other HTTP pages. For example, some web-based email services have code in every page that checks if certain cookies are set in the browser and if not, sends a request for the home page of the mail service to be loaded. This kind of code prevents the end user from viewing the captured page content. Also, the PickPacket user might not want the captured page to generate any HTTP requests unless he specifically clicks on one of the links on the page. Thus, we need a mechanism to block or prevent a captured page from automatically generating HTTP Requests.

Chapter 4

Reconstruction of Web-Based Email: The Solution

In the last chapter, we looked at the problems in reconstruction of web-based email traffic in PickPacket. In this chapter we present the solution developed for the same. The following terms are used in describing the solution.

- *HTTPMail* refers to web-based email. A HTTPMail connection is an HTTP connection that has a web-based email server as one of its end points.
- A *Mail Session* is the set of HTTPMail connections established between the web-based email server and client from the time an email user logged in to the time he logged out.

4.1 Design

In this section, we present the top level design for reconstruction of web-based email. We do this by describing the changes made to each of PickPacket's components.

- *PickPacket Configuration File Generator*: As described in the previous chapter, in order to capture web-based email traffic PickPacket has to be configured to capture all HTTP traffic that has any of the the web-based email server names as hostname. Thus the Configuration File Generator is changed to automatically include a list of hostnames belonging to web-based email servers in HTTP criteria.

- *PickPacket Filter*: Since web-based email traffic is HTTP, there is no change in the PickPacket Filter. The Filter works on the configuration file generated by the Configuration File Generator and captures all traffic originating or terminating at the web-based email servers.

- *PickPacket PostProcessor*: All the reconstruction logic lies in this component. The PostProcessor is provided with web-based email specific information via a configuration file *httpmail.cfg*. A sample configuration file can be found in the Appendix. The configuration file contains, among other things, a list of hostnames for each mail service. If the URL of an HTTP connection contains a hostname from this list, it means the connection belongs to the corresponding mail service. For example, an HTTP connection with the hostname mail.yahoo.com in its URL belongs to Yahoo.

  The PostProcessor essentially treats web-based email traffic as a special case of HTTP. When the PostProcessor processes an HTTP connection, it checks whether the URL of the connection contains the hostname of any of the web-based email services. If so, it marks the connection as belonging to "HTTP-Mail". The PostProcessor then processes the connection as a regular HTTP connection. At the end of its processing, it adds the connection to a hash. The client IP address of the connection and the mail service name are used as the hash key. After all connections in the Filter output file have been post-processed, the hashed connections are processed further from the point of view of web-based email reconstruction. After reconstruction, HTTPMail connections are written to a directory of their own and have HTTPMail instead of HTTP as the protocol name in the meta-information file.

- *PickPacket DataViewer*: The DataViewer treats HTTPMail as a separate protocol altogether. For each HTTPMail connection, the DataViewer displays a list of pages that belong to it. For each page it displays the type of page (i.e. Inbox, Mail, Compose etc), a link to the captured contents, the content type and the keywords found in the page.

4.2 Filtering Mail Sessions

As described earlier, PickPacket captures all connections that belong to any of the web-based email services. However, the PickPacket user is interested in only certain email addresses. Thus, we need to

- Group captured connections into Mail Sessions
- Filter out uninteresting Mail Sessions

The hash of HTTPMail connections created by the PostProcessor groups connections based on the the client IP address and mail service, as shown in Figure 4.1.

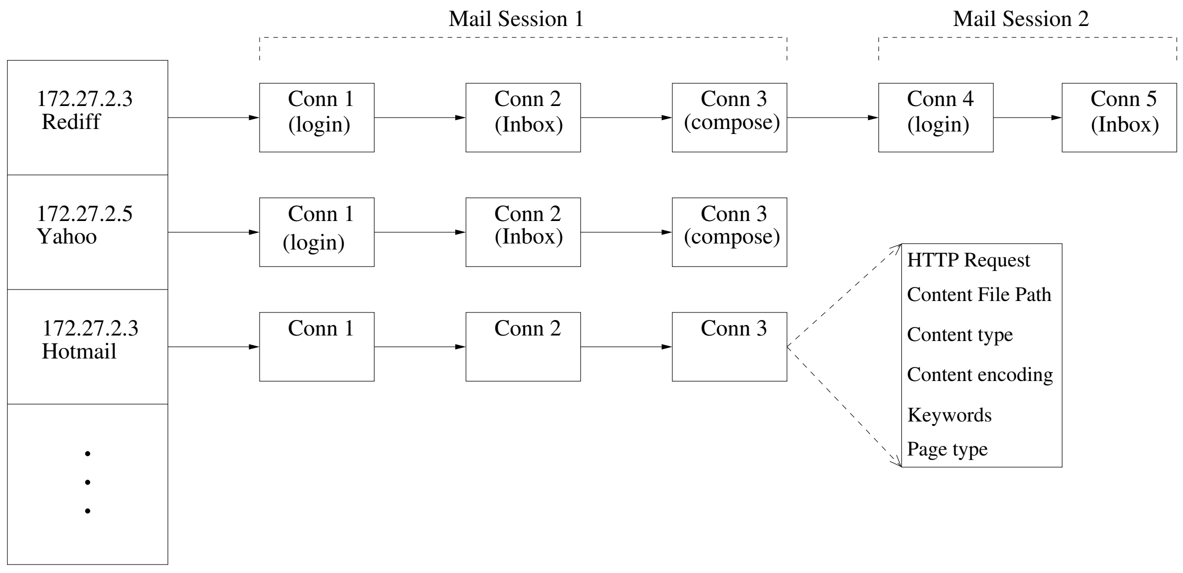


Figure 4.1: Hash of HTTPMail Connections

For each hash value, PickPacket maintains a linked list of HTTPMail connections. All connections in a list have the same destination IP and belong to the same mail service. Thus all connections in the list corresponding to a hash value can be grouped into a Mail Session provided we handle the case of multiple email users logging into the same email service from the same client machine (a common scenario in Internet Cafes). For example, consider the first entry in Figure 4.1. Here, the first three connections belong to one Mail Session while the next two belong to another. To separate such sessions, we need to be able to detect the HTTPMail connection that represents a new user logging in. The httpmail.cfg file lists the login URL for each email service. Thus, a connection that has the same URL as the login URL indicates the starting of a new Mail Session.

Since Pickpacket treats web-based email addresses as HTTP keywords, a Mail Session is of interest to PickPacket user only if at least one of its connections contains a keyword. Once we group Mail Sessions using the procedure discussed above, all Mail Sessions with no keywords can be filtered out.

4.3 Displaying Hidden Data

In this section, we describe the solution to the problem of reconstructing pages that accept text input from the user (described in Section 3.2). In order to describe the solution, we first take a look at some HTTP concepts including the different ways in which the text input entered by the email user can be sent to the HTTP server.

4.3.1 HTTP concepts

A *form* in HTTP is a container for other elements. An *input element* is an element that accepts input from the user. Some examples are text boxes, text areas, buttons and checkboxes. Every input element on an HTML page is normally a part of some form. The text entered by the user into an input element is sent to the server when the user *submits* the form containing the input element. Form submission in HTTP can be done using two methods - *GET* and *POST*. POST is used when the data that is being sent is likely to modify the state of the server. The content (data) being sent using POST can have the following content types depending on how they are encoded [10]:

1. *x-www-form-url-encoded*: This is the default encoding and is used when the

amount of data being sent is small or is in ASCII format. In case of web-based emails, it is used to send the username and password when the login page of a mail service is submitted. It is also used when the compose page is submitted to send the address, subject and content of mail. An example HTTP Request with POST data of content type x-www-form-url-encoded can be found in the Appendix.

2. *multipart/form-data*: This encoding is used while sending large amount of binary or non-ASCII data. This is especially used when uploading files to the HTTP server. In the context of web-based emails, it is used to send attachments to the server. An example HTTP Request with POST data of content type multipart/form-data can be found in the Appendix.

Thus, by looking at the content type of the POST data in an HTTP request, we can detect requests that submit some data to the server.

Another concept in HTTP is that of *referrers*. Referrer is an optional HTTP Request header that the client can use to specify the URL of the document that generated the HTTP request.

4.3.2 Finding the Request Generating Page

In order to display composed emails in PickPacket, we first need to find the HTTP request that sends email data to the server and the compose page that generated the HTTP Request. The procedure for this differs depending on whether the email is sent with or without attachments. We first look at the easier case of a mail sent without attachments.

1. Go through the hash of HTTPMail sessions and look for HTTP Requests with content type x-www-form-url-encoded.

2. If such an HTTP Request is found, it contains the POST data for a composed mail. Extract the referrer of the request.

3. Go through the entries in the current HTTPMail session and look for an entry with the same URL as the referrer found in the above step.

4. If such an entry is found, it represents the compose page that generated the HTTP Request found in Step 1.

While we have described the above procedure with respect to the compose page, the method remains the same if the POST data is generated by some other page, like the login page for example.

Now we look at the procedure for the more general case of a mail that may have been sent with attachments. To understand this procedure let us look at the actions that an email user performs in order to send a mail with attachment(s). He first opens the compose page and probably types in text. He then clicks on the "Add Attachment" link which opens a page on which he browses his filesystem and specifies the path of the attachment file(s). Once he has specified the path(s), the user clicks "Done" and comes back to the compose page. He then (optionally) edits the mail and sends it.

The sequence of HTML pages along with corresponding actions is shown in Figure 4.2

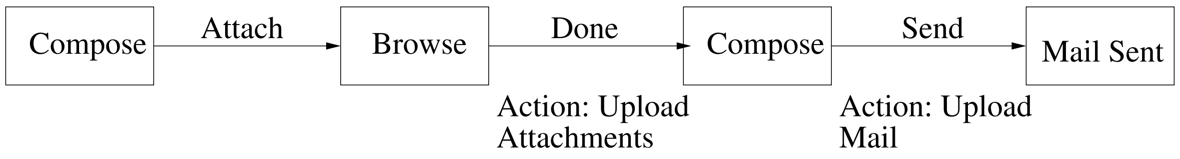


Figure 4.2: Composing an Email with Attachments

The HTTP Request to upload the (first) attachment contains the Browse page URL as its referrer which in turn contains the URL for the Compose page as referrer. Thus, going back two referrers from the first attachment gives us the URL of the compose page. We now present the procedure to find the Compose page that generated the HTTP POST requests for an email sent with attachments.

1. Go through the hash of HTTPMail sessions and look for HTTP Requests with content type either x-www-form-url-encoded or multipart/form-data.

2. If an HTTP Request with content type multipart/form-data is found, it contains attachment data. If this is the first attachment in this HTTPMail session, remember the referrer of the request.

3. Link the attachment found in the above step to other attachments found in this session. This is because a mail can have multiple attachments.

4. If an entry with content type x-www-form-url-encoded is found, it contains the POST email data (i.e., the addresses, subject and content of the mail). We now look for the Compose page that generated this data.

5. Go through the entries in the current HTTPMail session and look for an entry with the same URL as the referrer of the first attachment (found in step 2). We are looking for the entry corresponding to the Browse page.

6. If such an entry is found, extract the referrer of the HTTP Request for that entry.

7. Go through the entries in the current HTTPMail session again and look for an entry with the same URL as the referrer found in the step above. Now we look for the entry corresponding to the referrer of the Browse page, i.e., the Compose page.

8. If such an entry is found, it represents the Compose page for the mail.

4.3.3 Correlating POST Data with the Fields

Once we have the Compose page and the POST data generated by submitting the page, we need to populate the input fields in the Compose page with values from POST data. POST data of type x-www-form-url-encoded is basically a collection of name value pairs of the form

$$name_1 = value_1 \& name_2 = value_2 \& name_3 = value_3 \ldots$$

where $name_i$ is the name of an input field of the form submitted
$value_i$ is the value of that field.

To display a composed mail, we are interested in extracting values for To, Cc, Bcc, Subject and the mail body. However, since each web-based mail service has different names for these fields, we cannot scan the POST data looking for specific values of $name_i$. To develop a more general solution, we observe that to display a composed mail we only need to worry about fields of type input and text area. This is because the user can enter values only into these fields. JavaScript provides methods using which we can retrieve a list of all elements of a particular type on a page. Using this method, we can write code to fill up the compose page with values from the POST data using Algorithm 1.

Algorithm 1 Correlating POST values with Input fields

```
 1: Extract all name value pairs from the POST data into a list
 2: Get a list of all elements of type input and text area in the page
 3: for all elements $e$ do
 4:    if e.type != hidden then
 5:       for all name value pairs (n,v) do
 6:          if e.name == n then
 7:             e.value = v
 8:             break
 9:          end if
10:       end for
11:    end if
12: end for
```

The above algorithm is implemented using JavaScript and PHP and written into every Compose page found using the procedures described in Section 4.3.2. When the PickPacket user views the Compose page using the Data Viewer, this code gets executed (partly on the Data Viewer web server, partly on the browser) and the Compose page fields get filled with values from POST data.

4.4 Handling AJAX Based Services

As described in the previous chapter, AJAX based email servers do not transfer entire HTML pages to the client. Instead, the server simply sends data for the page. The AJAX engine on the client side is responsible for formatting the data and displaying it to the user. The data exchanged between the server and the client has no standard format and can be displayed only by the email service provider's own AJAX engine.

To reconstruct web pages that use AJAX in PickPacket, we first need to be able to detect pages that contain AJAX data. These pages can have content type "text/xml" or "text/html", so the content type does not help in detection. The approach used in PickPacket is to add the URL that refers to AJAX pages to the httpmail configuration file (httpmail.cfg). Any page whose URL contains the AJAX URL in httpmail.cfg contains AJAX data.

Once we have the page that contains AJAX data, we need to call the display engine for that email service to display the data. To be able to do this, we store a (suitably modified) copy of the display engine for each AJAX based mail service. When we come across a page that contains AJAX data, we insert a call to the display engine, passing the path of the captured file that contains AJAX data as a parameter. When the PickPacket user views an AJAX page using the Data Viewer, this inserted code that makes a call to the display engine get executed. The display engine formats the data and displays it.

4.5 Uninteresting Pages

In case of web-based email services, PickPacket captures all the data that is transferred between the email server and the client. This results in a lot of unnecessary pages being captured and displayed to the PickPacket user. For example, advertisement pages, the page that is displayed after an email is sent, the browse page used to upload attachments etc. In order to reduce the number of pages that the user has to go through, we have implemented a method for automatic classification of web-based email data.

4.5.1 Classifiers

One simple way of classifying the captured data is to use URLs. For each web-based email service, we can remember the URL for each interesting page. For example, the URL for an opened mail, an opened attachment, the compose page etc. However, this approach has several problems. Firstly, it is neither general nor immune to change. Secondly, in several mail services the same URL refers to different pages at different times. Thus, a more general approach is needed.

Naive Bayes classifiers are among the most successful and popular algorithms for classifying text documents. They have been very successfully used in filtering out unsolicited email or spam [4]. The conventional approach to automatic classification of text documents using naive Bayes classifier calculates how much each word that occurs in the document affects the class of the document. This calculation is carried out using the *training set*, which is a collection of documents along with their classification. The Top N words found using the above calculation are the *keywords* that are used to classify unknown samples. Generally the occurrence of a word is treated as a binary feature, i.e., the classifier only cares about whether a particular word occurs in a document or not. Lemmatizers can also be used to make the classifier more efficient by treating different forms of a word as the same word. We use the naive Bayes classifier implementation available from University of Magdeburg[15] to implement automatic classification of email pages in PickPacket.

4.5.2 Selecting Keywords

While the overall classification approach adopted in PickPacket follows the conventional method, there are several differences in the details. The training set in PickPacket is a collection of mail sessions belonging to different web-based email services. The training set thus consists of all HTTP connections established between the web-based email server and the email client from the time the email user logs in to the time he logs out. The procedure for selecting keywords in PickPacket varies from that used in the conventional approach. We discuss below our approach and the considerations that led to it.

1. Instead of considering each and every word that occurs in the training set, we start with a possible set of words that are most likely to affect the class of a document. This is possible in case of PickPacket because we know what the pages belonging to various classes look like and hence can guess what words will occur in those pages.

2. For most words, it is sufficient to check whether the word occurs in a document or not. In a few cases however, the number of occurrences of words gives a better estimate of the class. For example, consider the Inbox of an email user. For every mail displayed on the page, there is a checkbox by clicking on which the user can select the mail and then perform some action on the mail. While one or two checkboxes can occur in other pages too, if the count of checkboxes on a page exceed some threshold, it is extremely likely that the page is the Inbox of some user. Hence in this case, the number of occurrences of a particular word gives a better estimate of the class.

3. Another choice to be made is whether a word should be counted if it occurs as part of a bigger word. For example, if the keyword is "Send", should "Sender" be counted as an occurrence of Send? We found that the answer depends on the keyword in question. For most keywords, the accuracy of classification improves if only the occurrence of the entire word is counted. For a few words however, this is not the case. For example, the Inbox of any email service generally contains a link called ''prev" which can be used to view the previous set of messages. In some email services, this link is called "prev" while others use the whole word "previous". If we use "prev" as the keyword and count only entire occurrences of it, then we miss out on email services that use the word "previous". A few more examples of such words are "attach" and "spell".

Thus for each keyword, we have two features - whether the number of occurrences of that keyword need to be counted and whether only occurrences of the entire word need to be counted. After experimentation with the list of keywords and the values for these two features, we finalized on a list of 31 keywords. The keywords as well as the values for the above attribute are listed in httpmail.cfg, which can be found in the Appendix.

4.5.3 Target Classes

We define the following target classes into which Mail Session pages can be classified.

1. *Welcome*: This class consists of the first page that a web-based email user sees on logging in. It contains, among other email service specific things, the name of the user and the number of messages (read as well as unread) that are present in the user's Inbox.

2. *Inbox*: Pages that display a list of mails form this class. For example, the page that displays a list of all messages in the user's Inbox. Pages that display list of messages in any folder (whether pre-created by the web-based email service or created by the email user) fall into this category.

3. *Mail*: This class consists of pages that display an opened mail. Even mails that are loaded in the same page as the Inbox (as happens in the case of AJAX based mail services) form part of this class.

4. *Compose*: Pages in which the email user types in a mail form this class. Newly created mails, replies and forwards are included in this class.

5. *Other*: All other pages in a Mail Session are a part of this class. Some examples are - the page that gets displayed after the email user sends a mail stating that the mail has been sent, the Browse page that is displayed when the user uploads an attachment. Basically all pages that are not likely to be of interest to the PickPacket user are dumped into this class.

The keywords selected are such that the pages that fall into the *Other* class will have the least number of keywords.

4.5.4 Training the Classifier

In order to train a naive Bayes classifier to classify Mail Session pages, we represent each page in the training set by a vector $\langle X_1, X_2, X_3, \ldots X_{31}, C \rangle$ where each X_i

corresponds to one of the 31 keywords and C represents the target class of the page.

For keywords whose occurrences need not be counted,

$X_i = 0$ if the keyword does not occur in the page

$= 1$ if the keyword occurs in the page

For keywords whose occurrence needs to be counted,

$X_i =$ number of times the keyword occurs in the page

We call the vector corresponding to a page the *pattern* for the page.

We trained a naive Bayes Classifier by feeding it patterns for over 200 pages from Mail Sessions belonging to different email services. On test pages, we got a classification accuracy of over 90 percent. On analyzing the misclassification, we found that all of them involved pages belonging to the *Other* class. This is because while the remaining classes have well defined and distinguishable patterns, the *Other* class is a collection of widely varying patterns. This class is meant for all sorts of miscellaneous documents that don't have much in common except that they do not have too many of the keywords occurring in them. We use this fact to improve the classification accuracy. We split the classification process into two steps i.e. we train and use two naive Bayes classifiers. The first one simply classifies documents into two classes - *Useful* and *Not Useful*. Pages that belong to the *Other* class fall into *Not Useful* while all other pages are classified as *Useful*. The second classifier works only on pages in the *Useful* class and further classifies them into the remaining categories described above. The classification accuracy goes up remarkably using this method. We observed that only the following documents from our test set were misclassified.

- Welcome pages belonging to certain web-based email services. This is because the welcome page format varies widely across mail services. It is difficult to find keywords that are common to the page across email services.

- Folders with no messages. Checkbox count plays an important part in classifying a page as "Inbox". Since folders with no messages have a very low checkbox count, such pages are likely to be misclassified.

Since the misclassified pages are not likely to be of paramount importance to the PickPacket user, we decided not to tweak the classifier strings further.

4.5.5 Extracting Keywords

When the trained classifier is run as part of PickPacket, it needs to be supplied with the keyword vector corresponding to the page being classified. In order to search the page for keywords, some pattern matching algorithm needs to be used. We settled on the Aho-Corasick pattern matching algorithm [2] for the following reasons:

- It is among the fastest known algorithms for matching multiple patterns.
- It is recommended for use in scenarios where the same set of patterns is to be searched for in multiple texts.
- Since the strings to be searched for are not likely to change, the initial portion of the algorithm needs to be executed only once.

The Aho-Corasick algorithm works by constructing a finite state machine out of the patterns to be matched. Every state of the FSM has a name and an array of transitions. Elements of the array contain the next state for each input symbol. Certain states of the FSM also have outputs associated with them. Every such state indicates that some pattern(s) has matched. The FSM is applied to the input text in which the patterns are to be matched (Mail Session pages in our case). Whenever the FSM reaches a state which has an associated output, it means the corresponding pattern(s) has been found in the input text. The algorithm runs in O(n) time where n is the length of the input text and takes O(m) space in the worst case where m is the sum of the lengths of all the patterns.

In PickPacket, the FSM is constructed only once, the first time a Mail Session page needs to be classified. After that, the FSM resides in memory and is used for classification of remaining pages.

4.6 HTTP Requests From Captured Pages

Captured HTTP pages when viewed in a browser might generate HTTP requests. We need a solution that prevents such requests. One solution would be to replace all URLs in the captured page with local URLs. However, if this is done the hyperlinks on the page will no longer work. We want the solution to preserve hyperlinks since since hyperlinks do not generate HTTP requests unless the user clicks on them. Analysis of captured HTTP pages shows that HTTP Requests are generated for the following reasons:

1. To display an image
2. To include a page that contains client side scripting code
3. To include a stylesheet
4. To load a new page in the same window

The URL of the image, script or stylesheet can either be absolute or relative. When a captured page is viewed from the Data Viewer, relative URLs will not generate HTTP Requests to the the web-based email service provider's HTTP server. Hence we need to worry only about absolute URLs i.e. URL starting with the string "http://".

In order to stop HTTP Requests from being generated for the first three reasons listed above, we scan the entire page for the string "http://". If the string is found, we verify that it is not part of a hyperlink tag. We then replace the absolute URL by a URL relative to the path of the captured document. This relative URL points to a special folder called "img" which is created by the PostProcessor and into which the PostProcessor stores all images, scripts and stylesheet files captured by PickPacket Filter. Thus when the captured page is viewed in the Data Viewer all HTTP requests generated are to the Data Viewer's web server. If the image, script or stylesheet in the request has been captured by the Filter, it will be returned by the Data Viewer's web server.

Some web-based mail services load a new page in the same window if some conditions are not met. This not only generates an HTTP Request, but also prevents the PickPacket user from viewing the captured page. To prevent this, we scan the captured page for the pattern “window.open” and comment out the line where the pattern occurs. Note that replacing the URL of the page being loaded by a local URL will not really help. While this will prevent an HTTP request to the service provider's server from being generated, the PickPacket user will still not be able to view the captured contents, since the page corresponding to the replaced local URL will be loaded.

The Boyer Moore algorithm [5], implemented as part of PickPacket's string matching library, is used to search the captured pages for the two patterns discussed above. This algorithm is among the fastest known algorithms for single pattern search.

Chapter 5

Tests and Conclusion

In this chapter we first discuss how we have tested PickPacket after adding the new functionality. We then state our conclusions and provide pointers for future work.

5.1 Testing

Reconstruction of captured web-based email data was carried out for the following web-based email services:

1. Yahoo!
2. Rediffmail
3. Hotmail
4. Indiatimes

We captured Mail Sessions of email users belonging to each of the above email services. The captured data contained pages that represent a wide range of email user actions. For example sending and retrieving mails (with and without attachments), replying, forwarding and deleting mails, opening various mail folders, address book lookups, mail setting modifications. For each captured session, we verified the following:

- All pages belonging to the above email services are captured when the Filter is run with the configuration file generated using the modified Configuration File Generator.
- Mail sessions are correctly identified, especially in the case of multiple users logging in into the same mail service from the same client machine.
- Mail sessions that are not of interest to the PickPacket user are not displayed.
- The captured contents are visible when the PickPacket user views a page using the Data Viewer. This is specially important for pages belonging to mail services like Yahoo! which load the Yahoo Mail home page if certain cookies are not set in the browser.
- The captured page does not make any HTTP requests for image, script or stylesheet files to the website's HTTP server.
- When viewed in the Data Viewer, the input fields in Compose pages are filled with POST data from the corresponding HTTP Request.
- Pages belonging to AJAX based mail services are correctly displayed. Rediffmail pages were used for testing this.
- Mail Session pages are correctly classified.

In case of classification of captured data, we found that except for some *Welcome* pages and pages corresponding to empty folders, all other pages were correctly classified.

5.2 Conclusion

PickPacket is a very useful tool for capture and analysis of network traffic. It has a built-in support for a large number of application-level protocols which not only gives the PickPacket user more flexibility in specifying filtering criteria for those protocols but also makes it easier for the user to analyze data belonging to those protocols. In this thesis, we have attempted to extend this flexibility and ease of analysis to web-based email traffic. The PickPacket user can specify the web-based email addresses that he wishes to monitor and PickPacket will capture all mails sent to and from that mail address. This captured data is further processed so that the PickPacket user gets to see exactly what the email user must have seen on his screen, minus some images. Composed mails, sent and retrieved attachments, pages that contain AJAX data are all reconstructed and displayed in a much more user friendly and accessible format. We also classify the captured pages to indicate captured data that is likely to be of interest to the user. We have seen that this classification greatly reduces the amount of data that the PickPacket user has to go through. We have noted a reduction of 20 to 80 percent depending on the web-based email service in question.

5.3 Future Work

- Currently PickPacket reconstructs pages belonging to only one AJAX based email service (Rediffmail). The reconstruction logic can be extended to accommodate others, prominently Gmail.

- An email user might view the same page several times in a Mail Session. Currently PickPacket captures and displays the page each time the user views it. Some mechanism could be developed for detection of duplicate pages which will further reduce the amount of data that the PickPacket user has to analyze.

Bibliography

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

[2] Aho, A. V., and Corasick, M. J. Efficient string matching: an aid to bibliographic search. *Commun. ACM 18*, 6 (1975), 333–340.

[3] “AJAX: A new approach to web development”. http://adaptivepath.com/publications/essays/archives/000385.php.

[4] Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Sakkis, G., Spyropoulos, C. D., and Stamatopoulos, P. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach, 2000.

[5] Boyer, R. S., and Moore, J. S. A fast string searching algorithm. *Commun. ACM 20*, 10 (1977), 762–772.

[6] Degioanni, L., Risso, F., and Viano, P. “Windump”. http://netgroup-serv.polito.it/windump.

[7] et al., G. C. “Ethereal”. Available at http://www.ethereal.com.

[8] Graham, R. “carnivore faq”. http://www.robertgraham.com/pubs/carnivore-faq.html.

[9] “How Carnivore Works”. http://www.howstuffworks.com/carnivore.htm.

[10] "HTML 4.01 Specification ". http://www.w3.org/TR/html4/interact/forms.html.

[11] Jacobson, V., Leres, C., and McCanne, S. "tcpdump : A Network Monitoring and Packet Capturing Tool". Available via anonymous FTP from ftp://ftp.ee.lbl.gov and www.tcpdump.org.

[12] Jain, S. K. "Implementation of RADIUS Support in Pickpacket". Master's thesis, Department of Computer Science and Engineering, IIT Kanpur, Apr 2003. http://www.cse.iitk.ac.in/research/mtech2001/Y111122.html.

[13] Kapoor, N. "Design and Implementation of a Network Monitoring Tool". Master's thesis, Department of Computer Science and Engineering, IIT Kanpur, Apr 2002. http://www.cse.iitk.ac.in/research/mtech2000/Y011111.html.

[14] McCanne, S., and Jacobson, V. "The BSD Packet Filter: A New Architecture for User-level Packet Capture". In *Proceedings of USENIX Winter Conference* (San Diego, California, Jan 1993), pp. 259–269.

[15] "Full and Naive Bayes Classifiers". http://fuzzy.cs.uni-magdeburg.de/ borgelt/doc/bayes/bayes.html.

[16] Pande, B. "Design and Implementation of a Network Monitoring Tool". Master's thesis, Department of Computer Science and Engineering, IIT Kanpur, Sep 2002. http://www.cse.iitk.ac.in/research/mtech2000/Y011104.html.

[17] Smith, S. P., Jr., H. P., Krent, H., Mencik, S., Crider, J. A., Shyong, M., and Reynolds, L. L. "Independent Technical Review of the Carnivore System". Tech. rep., IIT Research Institute, Nov 2000. http://www.usdoj.gov/jmd/publications/carniv_entry.htm.

Appendix A

Sample Configuration File

A.1 Configuration File with Filtering Criteria (*http-mail.cfg*)

```
# Number of web based email services that PickPacket handles
Num_Of_Mail_Services=4

# Maximum number of HTTPMail connections stored per Mail Session
Max_httpmail_conns=100

# This section contains web based email service specific details.
# Service_Name: Name of the web based email service
# Login_Uri:  The URI used by email users to log into their web based
#   email account
# Logout_Uri: The URI that indicates that an email user has logged out
# of his account
# Num_Uri: Number of URI's that belong to this web based email service
# (apart from login and logout)
# Num_Ignore_Uri: Number of URIs to be ignored. In some cases, certain
# URIs are used only for doenloading images or scripts.
# PickPacket need not capture connections from these URIs.
# Num_Ajax_Mail_Uri: Number of URIs that correspond to pages containing
#  AJAX data
# Ajax_Mail_Filename: The name of the file that contains the (modified)
# AJAX display engine for the web based email service.

<Mail_Service>
Service_Name=Yahoo
Login_Uri=mail.yahoo.com
Logout_Uri=login.yahoo.com
Num_Uri=0
Num_Ignore_Uri=0
Num_Ajax_Mail_Uri=0
</Mail_Service>
<Mail_Service>
Service_Name=Rediff
Login_Uri=in.rediff.com
Logout_Uri=login.rediff.com
Num_Uri=2
Uri=mail.rediff.com
Uri=f2check.rediff.com
Num_Ignore_Uri=1
Uri=immail.rediff.com
Num_Ajax_Mail_Uri=1
Uri=bn/ajaxmail.cgi
Ajax_Mail_Filename=rediff_mail.html
</Mail_Service>
<Mail_Service>
Service_Name=Indiatimes
Login_Uri=in.indiatimes.com
Logout_Uri=integra.indiatimes.com
Num_Uri=1
```

```
Uri=infinite.indiatimes.com
Num_Ignore_Uri=1
Uri=favicon.ico
Num_Ajax_Mail_Uri=0
</Mail_Service>
<Mail_Service>
Service_Name=Hotmail
Login_Uri=login.passport.net
Logout_Uri=loginnet.passport.com
Num_Uri=2
Uri=hotmail.msn.com
Uri=^64.
Num_Ignore_Uri=0
Num_Ajax_Mail_Uri=0
</Mail_Service>

# List of patterns for classification of HTTPMail pages. Each pattern has two
# attributes:
# Count_Occurences: Whether the number of occurences of the pattern
#  need to be counted
# Match_Whole_Word: Whether only occurence of the entire pattern should
# be considered
<Pattern_List>
Num_Of_Strings=31
String=From
Count_Occurences=No
Match_Whole_Word=Yes
String=Sender
Count_Occurences=No
Match_Whole_Word=Yes
String=Subject
```

```
Count_Occurences=No
Match_Whole_Word=Yes
String=Date
Count_Occurences=No
Match_Whole_Word=Yes
String=Size
Count_Occurences=No
Match_Whole_Word=Yes
String=First
Count_Occurences=No
Match_Whole_Word=Yes
String=Prev
Count_Occurences=No
Match_Whole_Word=No
String=Next
Count_Occurences=No
Match_Whole_Word=Yes
String=Last
Count_Occurences=No
Match_Whole_Word=Yes
String=Delete
Count_Occurences=No
Match_Whole_Word=Yes
String=Reply
Count_Occurences=No
Match_Whole_Word=Yes
String=Reply All
Count_Occurences=No
Match_Whole_Word=Yes
String=Forward
Count_Occurences=No
```

```
Match_Whole_Word=Yes
String=To:
Count_Occurences=No
Match_Whole_Word=Yes
String=From:
Count_Occurences=No
Match_Whole_Word=Yes
String=Subject:
Count_Occurences=No
Match_Whole_Word=Yes
String=Date:
Count_Occurences=No
Match_Whole_Word=Yes
String=Sent:
Count_Occurences=No
Match_Whole_Word=Yes
String=prev
Count_Occurences=No
Match_Whole_Word=no
String=next
Count_Occurences=No
Match_Whole_Word=Yes
String=Cc:
Count_Occurences=No
Match_Whole_Word=Yes
String=Bcc:
Count_Occurences=No
Match_Whole_Word=Yes
String=Send
Count_Occurences=No
Match_Whole_Word=Yes
```

```
String=Cancel
Count_Occurences=No
Match_Whole_Word=Yes
String=Spell
Count_Occurences=No
Match_Whole_Word=No
String=Save
Count_Occurences=No
Match_Whole_Word=Yes
String=Draft
Count_Occurences=No
Match_Whole_Word=Yes
String=Welcome
Count_Occurences=No
Match_Whole_Word=Yes
String=checkbox
Count_Occurences=Yes
Match_Whole_Word=Yes
String=textarea
Count_Occurences=Yes
Match_Whole_Word=Yes
String=ttach
Count_Occurences=Yes
Match_Whole_Word=No
</Pattern_List>

# Filename of the naive bayes classifier that classifies pages into
# "‘Useful"’ and "‘Not Useful"’.
Basic_Classifier_Filename=basic_classifier.nbc

# Filename of the naive bayes classifier that further classifies "‘Useful"’
# pages
Advanced_Classifier_Filename=adv_classifier.nbc

# Name of the file containing names of columns of the classifier pattern
Classifier_Header_Filename=classify.hdr
```

Appendix B

HTTP Requests

B.1 Sample HTTP Request with ***x-www-form-url-encoded*** POST Data

```
POST http://us.f332.mail.yahoo.com/ym/Compose?YY=37103&order=down
&sort=date&pos=1 HTTP/1.1
Host: us.f332.mail.yahoo.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.2.1) Gecko/20030225
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;
q=0.2,text/css,*/*;q=0.1
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Proxy-Connection: keep-alive
Proxy-Authorization: Basic dmluYXlhbjpwYWFyY2gxMjM=
Referer: http://us.f332.mail.yahoo.com/ym/Compose?YY=18545
&order=down&sort=date&pos=1
Cookie: B=bkutevt1r71ij&b=2;F=a=xwKo...
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 441

SEND=&SD=&SC=&CAN=&docCharset=iso-8859-1&PhotoMailUser=
&PhotoToolInstall=&OpenInsertPhoto=&PhotoGetStart=0&
SaveCopy=yes&PhotoMailInstallOrigin=&.crumb=3rzjJYND0cL
&box=&FwdFile=&FwdMsg=&FwdSubj=&FwdInline=&OriginalFrom=
&OriginalSubject=&InReplyTo=&NumAtt=0&AttData=&UplData=
&OldAttData=&OldUplData=&FName=&ATT=1&VID=&showcc=&showbcc=
&AC_Done=&AC_ToList=&AC_CcList=&AC_BccList=
&To=canjali%40iitk.ac.in&Cc=&Bcc=&Subj=&Body=This+is+the+mail+body%2
```

B.2 Sample HTTP Request with *multipart/form-data* POST Data

```
POST http://by22fd.bay22.hotmail.msn.com/cgi-bin/doattach HTTP/1.1
Host: by22fd.bay22.hotmail.msn.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.2.1) Gecko/20030225
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;
q=0.2,text/css,*/*;q=0.1
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Proxy-Connection: keep-alive
Proxy-Authorization: Basic dmluYXlhbjpwYWFyY2gxMjM=
Referer: http://by22fd.bay22.hotmail.msn.com/cgi-bin/premail
Cookie: MSNADS=UM=; HMSatchmo=0; MSPAuth=7rbcpEp6IF3ZShDZLjIDi6...

Content-Type: multipart/form-data;
boundary=---------------------------10846777401547433821321324151
```

```
Content-Length: 3695

-----------------------------108467774015474338213213241 51

Content-Disposition: form-data; name=""

-----------------------------108467774015474338213213241 51

Content-Disposition: form-data; name="curmbox"

F000000001

-----------------------------108467774015474338213213241 51

Content-Disposition: form-data; name="HrsTest"

-----------------------------108467774015474338213213241 51

Content-Disposition: form-data; name="a"

4635ee47b81fedf232a638e4fc0047c91455023bf0173d9545bca793553e019d

-----------------------------108467774015474338213213241 51

Content-Disposition: form-data; name="attachmentfilename"

-----------------------------108467774015474338213213241 51

Content-Disposition: form-data; name="attachorcancel"

-----------------------------108467774015474338213213241 51
```

```
Content-Disposition: form-data; name="userfilename"

-----------------------------108467774015474338213213241 51

Content-Disposition: form-data; name="contentType"

-----------------------------1084677740154743382132132415 1

Content-Disposition: form-data; name="smsg"

ddk_2005.saved

-----------------------------10846777401547433821321324151

Content-Disposition: form-data; name="attfiles"

-----------------------------10846777401547433821321324151

Content-Disposition: form-data; name="attlistfile"

-----------------------------10846777401547433821321324151

Content-Disposition: form-data; name="badattfiles"

-----------------------------10846777401547433821321324151

Content-Disposition: form-data; name="ref"

-----------------------------10846777401547433821321324151
```

```
Content-Disposition: form-data; name="RTEbgcolor"

-----------------------------108467774015474338213213224151

Content-Disposition: form-data; name="from"

compose

-----------------------------108467774015474338213213224151

Content-Disposition: form-data; name="subject"

About an idiot

-----------------------------108467774015474338213213224151

Content-Disposition: form-data; name="oldattfile"

-----------------------------108467774015474338213213224151

Content-Disposition: form-data; name="_HMAction"

-----------------------------108467774015474338213213224151

Content-Disposition: form-data; name="attfile"; filename="dump.gif"

Content-Type: image/gif

GIF89a8

<GIF data ...>
```