# Firewall Log filter.

Jiang, Peng

**Publisher's version / Version de l'éditeur:**
https://doi.org/10.4224/18253434
*Student Report; no. SR-2009-20, 2009-01-01*

National Research Council Canada
Institute for Ocean Technology

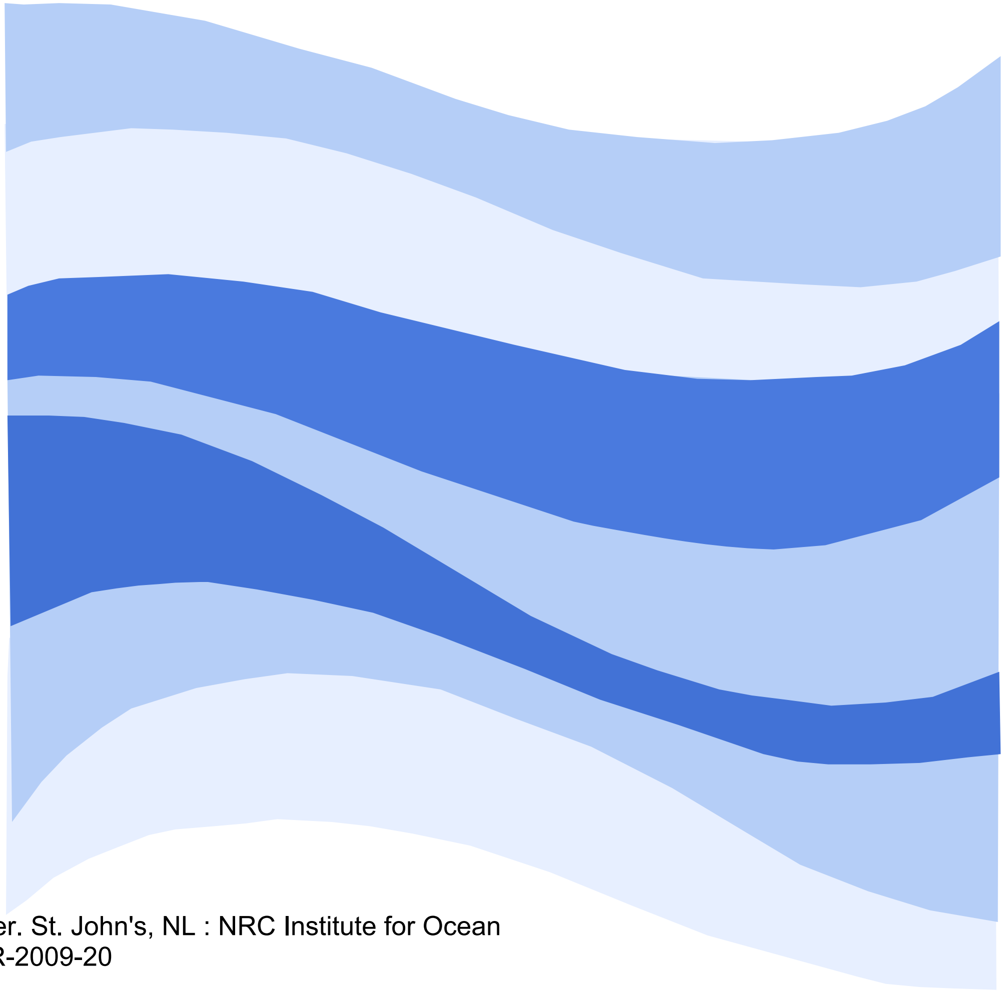
Conseil national de recherches Canada
Institut des technologies océaniques

SR-2009-20

Student Report

# Firewall Log filter.

Jiang, P.



Jiang, P., 2009. Firewall Log filter. St. John's, NL : NRC Institute for Ocean Technology. Student Report, SR-2009-20

# DOCUMENTATION PAGE

| REPORT NUMBER | NRC REPORT NUMBER | DATE |
|---|---|---|
| SR-2009-20 |  | December 2009 |

| REPORT SECURITY CLASSIFICATION | DISTRIBUTION |
|---|---|
| Unclassified | Unlimited |

**TITLE**

**FIREWALL LOG FILTER**

**AUTHOR(S)**

Peng Jiang

**CORPORATE AUTHOR(S)/PERFORMING AGENCY(S)**

National Research Council, Institute for Ocean Technology, St. John’s, NL

**PUBLICATION**

**SPONSORING AGENCY(S)**

| IOT PROJECT NUMBER | NRC FILE NUMBER |
|---|---|
|  |  |

| KEY WORDS | PAGES | FIGS. | TABLES |
|---|---|---|---|
| Firewalls, Networks, Security, Logs, OpenBSD, PF Filter | 27,<br>App. A-G | 4 |  |

**SUMMARY**

The need for Firewall Log Filter. In many cases, it is possible to detect patterns by browsing the log data but unfortunately it is also tedious. For example, a clever attack against a firewall cluster of an enterprise is scattered over all of its firewalls and executed slowly from several different IP addresses using all the possible protocols alternately. In such situation, we have to use the log filter to collect the correlated IP addresses. The typical size of the firewall log entries was more than 100,000 lines, which were collected during a period of a day. From these entries, with the frequency of equal or greater than 5,000 the FLF was able to identify the pattern and was able to generate a summary. When the frequency was lowered to 50, the FLF also has the ability to ignore generating summaries in order to save computation and analyzing time.

**ADDRESS**

National Research Council
Institute for Ocean Technology
Arctic Avenue, P. O. Box 12093
St. John's, NL A1B 3T5
Tel.: (709) 772-5185, Fax: (709) 772-2462

National Research Council
Canada

Institute for Ocean
Technology

Conseil national de recherches
Canada

Institut des technologies
océaniques

# Firewall Log Filter

SR-2009-20

Peng Jiang

December 2009

# Executive Summary

Firewall Log Filter (FLF) is a scripting application, where network packets were captured from the network interface controller by means of "tcpdump"; then FLF could parse, filter and summarise the logged entries without losing any important information. By combining log entries with their frequencies and identifying recurring patterns, FLF was able to separate correlating entries from infrequent ones and display them with accompanying information. Thus, the administrators can have a more predicted outlook of the logged entries and can identify suspicious network activities. For example, Administrators can trace the source IP addresses by means of programs such as "nslookup", "whois", "traceroute", etc.

The need for Firewall Log Filter was also the motivation of this project. In many cases, it is possible to detect patterns by browsing the log data but unfortunately it is also tedious. For example, a clever attack against a firewall cluster of an enterprise is scattered over all of its firewalls and executed slowly from several different IP addresses using all the possible protocols alternately. In such situation, it can use the log filter to collect the correlated IP addresses.

The firewall logs contain information about the IOT's network traffic flow. The log entries provide a non-continuous flow of data, which means, it is not occasional bursts. If there are fragmented log entries that escaped from the firewall listening device, the FLF is able to either ignore those entries or list them for post analysis.

The typical size of the firewall log entries were more than 100, 000 lines, which were collected during a period of per day. From these log entries, with the frequency of equal or greater than 5, 000, FLF was able to identify the pattern and was able to generate a summary. When the frequency was lowered to 50, the FLF also has the ability to ignore generating summaries in order to save computation and analyzing time.

# Table of Contents

## List of Figures

# 1.0 Introduction

Firewall Log Filter (FLF) is a log filtering application developed for use at the Institute for Ocean Technology (IOT). The motivation for the development of a firewall-filtering program is to filter the firewall logging information, which was captured from the pf (packet filter) device. Then it is desirable to generate a report to show the network traffic activities. Firewall Log Filter currently has one main function, which is to generate a list of source IP address that is sorted in descending order by frequency.

## 1.1 Networks

If the 18th Century was the time of the great mechanical systems accompanying the Industrial Revolution, and if the 19th Century was the age of the steam engine, and then the key technology for the 20th Century has been information gathering, processing, and distribution. The question then came up: " How is information being shared and distributed?" Networks. [1]

A computer network is a simply two or more computers connected together so that they can exchange information. A small network can be as simple as two computers linked together by a single crossover cable as shown in Fig 1:

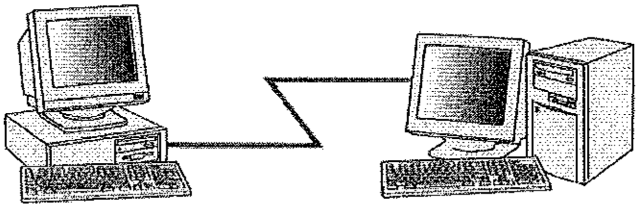


**Figure1 Simple Network**

A more complex network is IOT's, which is composed of multiple routers, switches, firewalls, and other network devices. Figure 2 shows the simplified network configuration.

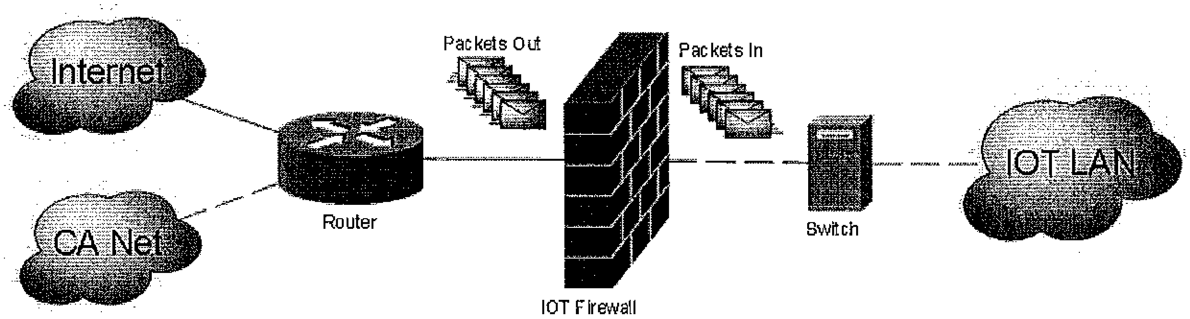


**Figure 2 IOT Network**

## 1.2 Firewalls

NRC-IOT uses Packet Filter as its firewall. It is easier to set up and configure, more flexible and efficient to implement. Furthermore, the firewall failover is another important reason why OpenBSD Packet Filter or "BPF" has been chosen for the implementation.

It is necessary to have redundancy that the OpenBSD offers. For example, IOT's firewall machines, one of the hosts becomes "Master" of a pre-assigned IP address, while the other will go into "Backup" mode. If the master machine fails to operate after a certain period of time, then the backup machine will automatically assume the master is down and finally take over as master to operate.

A firewall is a part of a computer system or network that is designed to block unauthorized access while permitting authorized communications. It is a device or set of devices configured to permit, deny, encrypt, decrypt, or proxy all (in and out) computer traffic between different security domains based upon a set of rules. Firewalls can be implemented in either hardware or software, or a combination of both.

From the above definition of a firewall, firewalls can be summarised as frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets. All traffic entering or leaving the intranet must pass through the firewall. The firewall examines each message and blocks those that do not meet the specified security criteria. It acts like a gate to protect assets, to ensure that nothing private goes out and nothing malicious comes in. Figure 2 shows the placement of a firewall.

There are several techniques of firewalling: Packet filter, Application gateways, Circuit-level gateway and proxy servers.

Packet filtering inspects each packet passing through the network and accepts or rejects it based on user-defined rules. Although it is difficult to configure, it is fairly effective and mostly transparent to its users. Packet filters can also be implemented as network layer firewalls, because they operate at a relatively low level of the TCP/IP protocol stack. Most often, the network administrators define the rules, as sometimes using the default rules are not safe enough. Packet filter techniques can be further divided into two categories, stateful and stateless. The difference between those two is that stateful technique maintains context about active sessions and uses that state information to speed up the process of packet filtering.

For example, IOT's network connection can be described by several properties from the BPF logging fields, including source and destination IP address, UDP or TCP ports, and current stage of the connection's TTL (Time To Live), which includes session initiation, handshaking, data transfer, or completion connection. That means, if a packet doesn't match an existing connection, it will be evaluated based on the rule-sets for new connections. However, if it matches an existing connection based on comparison with the firewall's state table, then it will be allowed to pass without further processing. On the other hand, stateless firewall requires lower system memory and can be faster for simple packet filters that require less time to filter than to look up an entire session.

Application gateway applies security mechanisms to specific applications, such as FTP and Telnet servers. This is very effective, but can create performance degradation. In essence, application gateways can prevent all unwanted outside traffic from reaching protected machines.

Circuit-level gateway applies security mechanisms when a TCP or UDP connection is established. Once the connection has been made, packets can flow between the hosts without further checking. Proxy server intercepts all messages entering and leaving the network. The proxy server effectively hides the true network addresses.

The major difference between ALG (Application Level Gateways) and Circuit-level Gateways (CLG) is the way they handle information. While a circuit level gateway only examines the address and port information contained in data it receives, not the content, the ALG is more in-depth of handling data. A firewall using ALG runs proxy applications to view common types of data before it is allowed through the firewall, for examples, web-pages, FTP, SMTP or POP3 for email, etc.

## 1.3 OpenBSD and BPF

A number of questions may come to mind from the reading of the last section: What is OpenBSD, why had that operating system been chosen for IOT's firewall system, what is the relation between OpenBSD and BPF, as well as what is BPF and how has it operate inside of the OpenBSD? OpenBSD is a free, multi-platform BSD-based Unix-like operating system. The main reasons that IOT's computer systems group uses OpenBSD are: it provides fail-over function in order to double secure the system and it runs on many different hardware platforms, for example, the alpha, the amd64, and the hppa, etc. It integrates cutting-edge security technology suitable for building firewalls and intranet like IOT's network. It provides the best development platform for building firewalls. It pays lots of attention to security problems and fixes.

OpenBSD's Packet Filter is OpenBSD's firewall system for filtering TCP/IP traffic and doing Network Address Translation. BPF under OpenBSD is also capable of normalizing and conditioning TCP/IP traffic and providing bandwidth control and packet prioritization. Additionally, OpenBSD also offers tools like Tcpdump for user-level packet capture and makes it possible the use of generic purpose of network monitoring. BPF, as the internal firewall of the OpenBSD, has two main components: the network tap and the packet filter. The network tap collects copies of packets from the network device drivers and delivers them to a listening program. The filter in the firewall then decides if a packet should be accepted or not according to rule sets and it also determines how many packets are going to be copied to a listening program.

Figure 3 shows an overview of BPF [2]. In the figure, it illustrates the BPF's interface with the rest of the system. When a packet arrives at a network interface the link level device driver normally sends it up the system protocol stack. And when the BPF is listening on this interface, the driver first calls BPF and it feeds the packet to each participating process's filter. For each filter that accepts the packet, BPF copies the requested amount of data to buffer associated with that filter. Then, the device regains the control again. It operates and listens over and over like the picture shows. All in all, the combination of OpenBSD and its BPF is the most secure and powerful operating system that packet filter can run on. And that's why IOT chose that system to protect and operate its network.

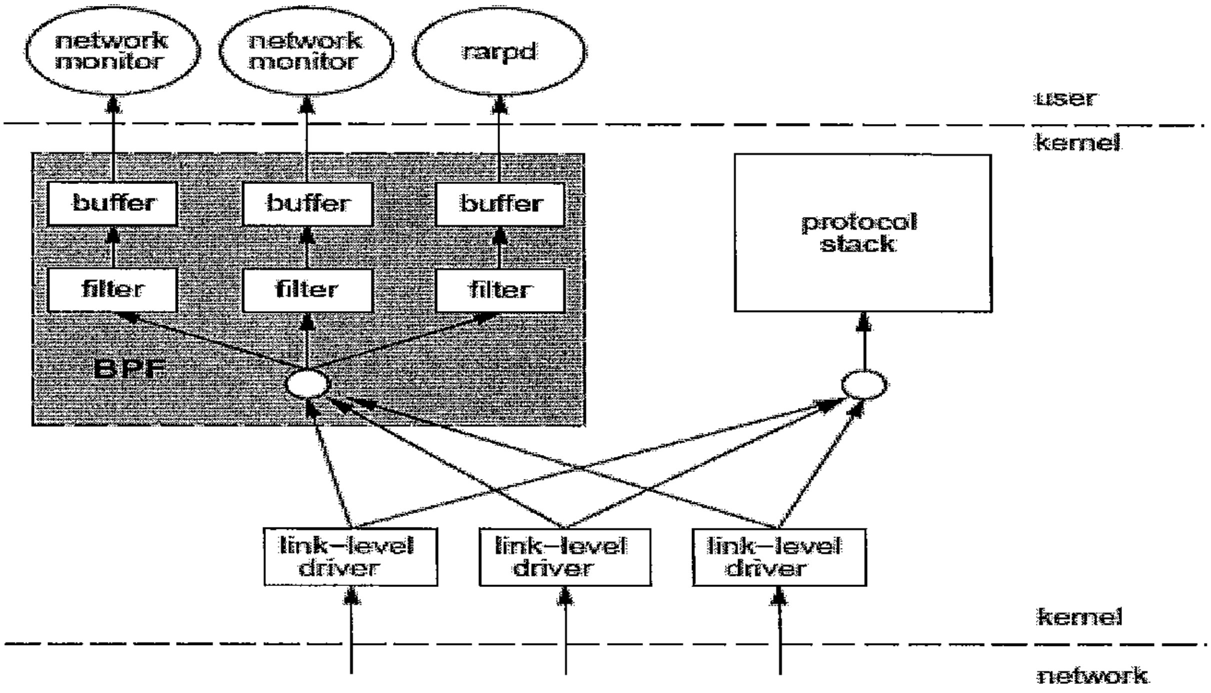


Figure 3 Overview of BPF

## 1.4 BPF Logging

BPF log entry is a snapshot of the packet that was either passed or blocked by the BPF rule-sets. The log files filtered by the rules could be the net flow of the network traffic, such as the source and destination IP, protocols information. Normally BPF is enabled when firewall machine is booted, and then there is an interface called pflogd, which logs traffic and firewall actions, like Tcpdump, to examine BPF's logging data from the kernel. When a packet is logged by PF, a copy of the packet header is sent to a pflog interface along with some additional data such as the interface the packet was transmitting, the action that BPF takes (pass or block), etc. By default, the pflogd daemon listens on the pflog1 interface and writes all logged data to the /var/log/pflog file. There are two ways of dumping the logging information to the output, which are reading and parsing from the Linux console and redirecting the output to a log file. From the log file, you will find different fields, for example, source and destination IP address, UDP or TCP ports, and current stage of the connection's TTL. The fields that are listed are typical outputs of Tcpdump, but the administrators can choose different flags and will get different outputs.

The log file written by pflogd is in binary format and must be decoded. It is necessary to use a program called, Tcpdump, to sort this out.

This is how to dump the traffic to a human-readable text file: on any console terminal type “tcpdump –netttr /var/log/pflog”. These ASCII format firewall logs are then sent to a remote logging server for further information processing.

# 2.0 Code Development

## 2.1 Design Overview

Firewall Log Filter was developed to be able to parse the log data and finally to generate a report to show the simple statistics of the data. These statistics can be used to find what is the most popular source IP address in the PF log. That is, why there should be two stages of project development. The first stage would be for opening the files, while the second stage would be for processing this data. Then questions arose regarding what to use to open the log files, how to parse the data, and which form of the data would be processed? Additionally, how secure and efficient was opening and processing the log data. With these questions in mind, it is worthwhile to consider what would be necessary in developing new code for the first stage or to use an open source program like Tcpdump or Wireshark. While the answer was obvious, it is still worth investigating because in order to open the encoded raw binary pcap format log files, Tcpdump has to use libpcap library as library support.

It would be convenient and useful to find an Application Programming Interface like Pcapy, which is a Python extension module interface with the libpcap packet capture library, as a wrapper to implement the underlining libpcap functions, pcap_open_offline as one of the most important functions used to open the offline pcap files.

## 2.2 Tcpdump

For the current project stage, it is necessary to use Tcpdump as an intermediate method to open the binary log files to convert to ASCII based text files so that it would be easier to parse the information. Before touching any log entries, it was necessary to know how to use Tcpdump to open and read log files. Tcpdump prints out the headers of packets on a network interface that match the Boolean expression.

## 2.3 The Firewall Log Filter

Firewall Log Filter is a Python script program for use at IOT. Python was used because it is platform independent and is dynamic and efficient. Additionally, it is also a standard language for software development at NRC-IOT. Most importantly, it is easier to learn and write while being used in a wide variety of application. Parsing the contents of Tcpdump log-files in /var/log/ is probably not the most efficient way of converting data, however, it is a very simple and easy-to-code approach.

## 2.4 Flowchart of the Code

Figure 4 shows the logic of the parsing function in the script.

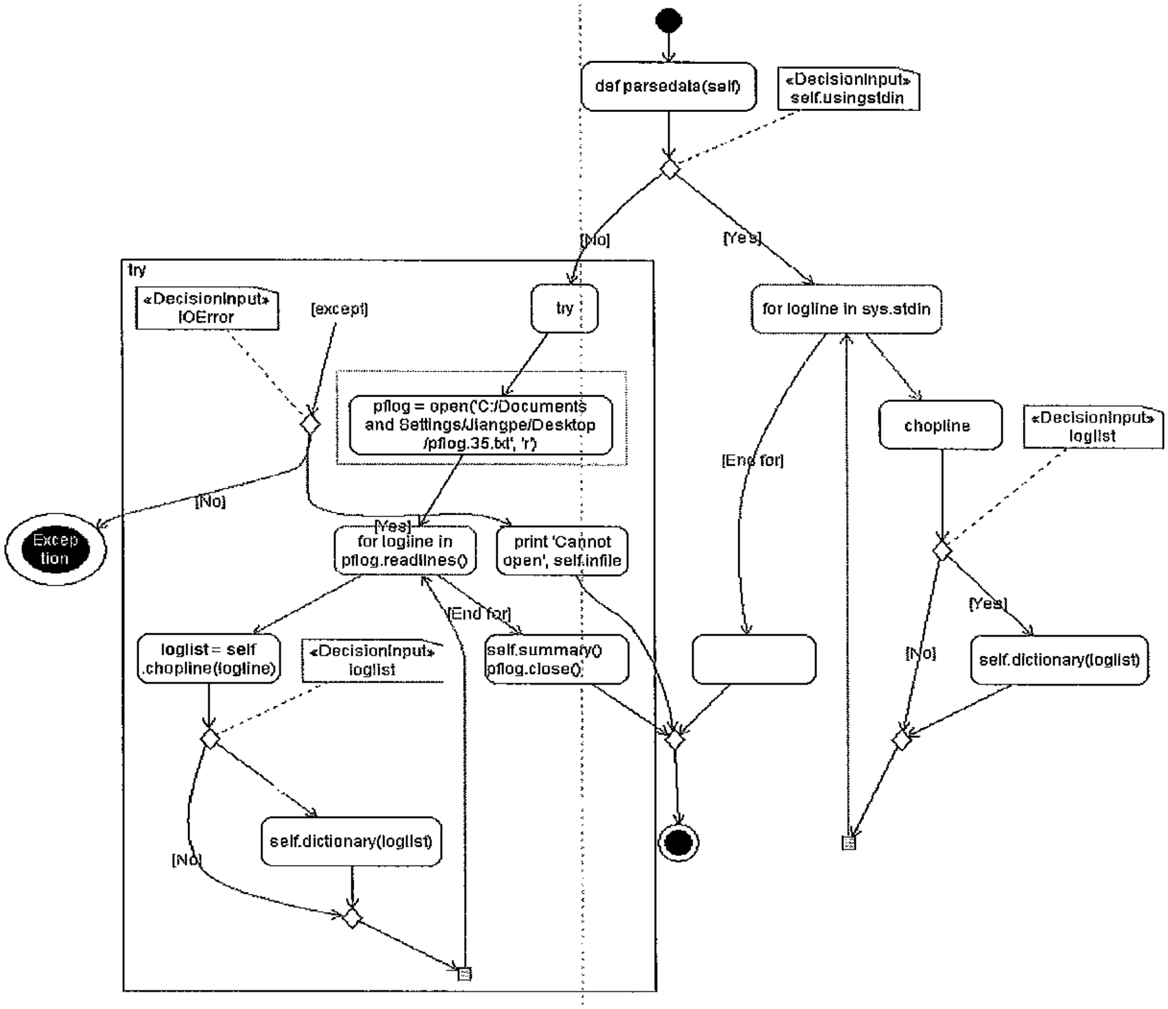


**Figure 4 OOP Class Method Parsedata Code Flow chart**

### 2.3.2 Usage

The current stage of development of Firewall Log Filter program evaluates basic statistic summary. It offers a preliminary review and general statistics of BPF logs. Further development of the Firewall Log Filter program will be much easier based on the code, because it was programmed using OOP (Objective Oriented Programming) class, which had has been developed methods such as: "parsedata", "chopline", "dictionary", "summary", and "the_main_function". Figure

If you're under Windows, which means, you have already had the text version of the log files, in this case, you don't need to use Tcpdump, but you have to point to where the log files locates on Windows. Open up Python shell and run BPFparse.py under windows.

## 4.0 Conclusions

Network traffic are filtered by the BPF and logged. The log files needed to be parsed so that administrators can review the traffic patterns and then predict the trends of the network traffic while analyzing packets activities for auditing the network activities. As mentioned in the previous sections about what and why it is needed to monitor and filter firewall log files and what is the best and efficient way of predicting network traffic and any suspicious activities, it is necessary to have a filter program for getting the particular statistics of the network activities Firewall Log Filter script came up with the idea of getting the frequency of the most popular source IP address so that administrators can monitor and manage the network more efficiently.

## 5.0 Recommendations

In order to efficiently audit and manage the network packet activities, firewall log filtering plays a very important role in network management. It is recommended that network monitoring and logging automation of every step would save time and human resources. Therefore, it is recommended to have a log file server that is exclusively targeted to manage and monitor log files. This can be accomplished by using standard applications. Further recommendations would fully develop the Firewall Log Filter program; for example, add more functions to the existing code to give a GUI interface for user-friendly interactions.

# 6.0 References

[1] Andrew S. Tanenbaum, Computer Networks 3rd ed. Prentice Hall PTR Upper Saddle River, New Jersey 1996

[2] Steven McCanne and Van Jacobson (1992, December 19). The BSD Packet Filter: A New Architecture for User-Level Packet Capture. Lawrence Berkeley Laboratory. Retrieved from http://www.tcpdump.org/papers/bpf-usenix93.pdf

[3] OpenBSD software team (1999, October 20) PF: The OpenBSD Packet Filter. Retrieved from http://www.openbsd.org/faq/pf/logging.html

[4] Wikipedia team. Science and Technology. (2009, December). Retrieved from http://en.wikipedia.org/

[5] Tcpdump software team (2001, September 30). PCAP Manual Page. Retrieved from http://www.tcpdump.org

[6] Python docs team. Documentation. Retrieved from http://python.org/doc

[7] Tcpdump software team (2001, September 20). BPF Usenix. Retrieved from http://www.tcpdump.org/papers/bpf-usenix93.pdf

[8] Wong, Gilbert. Computer Services Group. Institute for Ocean Technology. St. John's, NL, Canada. Personal Correspondence. 2009.

[9] Walsh, Doug. Computer Services Group. Institute for Ocean Technology. St. John's, NL, Canada. Personal Correspondence. 2009.

[10] Elms, Wayne. Computer Services Group. Institute for Ocean Technology. St. John's, NL, Canada. Personal Correspondence. 2009.

[11] Lutz, Mark and David Ascher. Learning Python. 2nd ed. Sebastopol, CA, United States: O'Reilly Media, 2003.

# Appendix A – Firewall Log Filter Source Code

Note: The code in these appendices was not formatted very nicely by the word processor. For a serious look at the code one should load the files into a code editor (Python IDEL works well).

MAIN.PY

```
import sys
import os
import re
import string
import time

class BPFfilter:

    def __init__(self):
        self.pfdict = {}
        self.usingstdin = False
        self.infile = ''

    def parsedata(self):
        if self.usingstdin:
            for logline in sys.stdin:
                loglist = self.chopline(logline)
                if loglist:
                    self.dictionary(loglist)
            self.summary()
        else:
            try:
```

```
        pflog = open('C:/Documents and
Settings/Jiangpe/Desktop/pflog.35.txt', 'r')#Give the location of log file on your
system.
      except IOError:
        print 'Cannot open', self.infile
      else:
        for logline in pflog.readlines():
          loglist = self.chopline(logline)
          if loglist:
            self.dictionary(loglist)
        self.summary()
        pflog.close()

  def chopline(self, logline):
    pattern = re.compile('\s(block).*:\s(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,5})\.*')
#Change 'pass' to 'block' or vice versa will generate pass or block with sorted
port numbers.
    #pattern = re.compile('\s(block).*:\s(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.*')#This
gives you 'pass' or 'block' srcip without port numbers.
    if not pattern.search(logline):
      return []
    logtuple = pattern.search(logline).groups()
    loglist = list(logtuple)
    return loglist

  def dictionary(self, loglist):
    netflow = loglist[0]
    srcip   = loglist[1]
    if not netflow in self.pfdict:
      self.pfdict[netflow] = {}
    if not srcip in self.pfdict[netflow]:
            self.pfdict[netflow][srcip] = 0
        self.pfdict[netflow][srcip] = self.pfdict[netflow][srcip] + 1

    def summary(self):
        print '------------------------------'
        print 'List of pfLog SrcIP Activities:'
        print '------------------------------'
        for netflow in self.pfdict:
            print netflow
            print '                              Total of',len(self.pfdict[netflow]), 'scrip(non-
repeats);'
            srclist = self.pfdict[netflow].keys()
        getmode = []
        for srcip in self.pfdict[netflow]:
            getmode.append((self.pfdict[netflow][srcip], srcip))
        getmode.sort()
        getmode.reverse()
        for x in getmode:
            print "%3d : %s" % (x[0], x[1])
if __name__ == '__main__':
    logobj = BPFfilter()
    logobj.parsedata()
```

# Appendix B –API Code

Note 1: The code in these appendices was not formatted very nicely by the word processor. For a serious look at the code one should load the files into a code editor.

Note 2: The code below is for the first stage of development of the Firewall Log Filter Program. It is partially completed. But it was worth the effort to further dig into it. The reference can be found from the Reference List entry 12.

```
#include <stdio.h>
#include <stdlib.h>
#include <pcap.h>  /* GIMME a libpcap plz! */
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>


/*
 * gcc ttt.c -lpcap
 */
int
main( int ac, char *av[] )
{
		pcap_t *handle;				/* Session handle */
		char *dev;			/* The device to sniff on */
		char errbuf[PCAP_ERRBUF_SIZE];		/* Error string */
		struct bpf_program fp;			/* The compiled filter */
		char filter_exp[] = "port 23";	/* The filter expression */
		bpf_u_int32 mask;		/* Our netmask */
		bpf_u_int32 net;		/* Our IP */
		struct pcap_pkthdr header;	/* The header that pcap gives us */
		const u_char *packet;			/* The actual packet */
```

```
                /* Define the device */
                dev = pcap_lookupdev(errbuf);
                if (dev == NULL) {
                        fprintf(stderr, "Couldn't find default device: %s\n", errbuf);
                        return(2);
                }
                /* Find the properties for the device */
                if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
                        fprintf(stderr, "Couldn't get netmask for device %s: %s\n", dev,
errbuf);
                        net = 0;
                        mask = 0;
                }
                /* Open the session in promiscuous mode */
                handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
                if (handle == NULL) {
                        fprintf(stderr, "Couldn't open device %s: %s\n", somedev, errbuf);
                        return(2);
                }
                /* Compile and apply the filter */
                if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
                        fprintf(stderr, "Couldn't parse filter %s: %s\n", filter_exp,
pcap_geterr(handle));
                        return(2);
                }
                if (pcap_setfilter(handle, &fp) == -1) {
                        fprintf(stderr, "Couldn't install filter %s: %s\n", filter_exp,
pcap_geterr(handle));
                        return(2);
                }
                /* Grab a packet */
                packet = pcap_next(handle, &header);
                /* Print its length */
                printf("Jacked a packet with length of [%d]\n", header.len);
                /* And close the session */
                pcap_close(handle);
```

```
                    return(0);

   char errbuf[PCAP_ERRBUF_SIZE];
   struct bpf_program bpfprog;
   pcap_t* pt;
   int status;

   pt = pcap_open_offline("pflog", errbuf);
   status = pcap_compile( pt, &bpfprog, "tcp", 1, 0);
   if ( status ) {
      printf("err %s\n", pcap_geterr(pt));
   }
   return 0;
}
```

# Appendix C – Typical PF Log Excerpt

```
15:00:02.170620 rule 0/(match) block in on em0: 220.163.158.217.17972 >
192.75.14.246.60522: udp 104
15:00:02.209833 rule 0/(match) block in on em0: 89.27.236.16.11846 >
192.75.14.246.55491: udp 103
15:00:02.353265 rule 0/(match) block in on em0: 116.11.159.217.7684 >
192.75.14.246.53289: udp 104
15:00:02.387372 rule 0/(match) block in on em0: 125.110.192.188.57917 >
192.75.14.246.56294: udp 104
15:00:02.396973 rule 0/(match) block in on em0: 125.110.192.188.57917 >
192.75.14.246.56294: udp 227
15:00:44.654377 rule 20/(ip-option) pass in on em2: 10.20.0.1 >
239.255.255.250: igmp nreport 239.255.255.250 [ttl 1]
15:00:44.654387 rule 20/(ip-option) pass in on em2: 10.20.0.1 >
239.255.255.250: igmp nreport 239.255.255.250 [ttl 1]
15:00:44.658237 rule 20/(ip-option) pass in on em2: 10.20.1.2 >
239.255.255.250: igmp nreport 239.255.255.250 [ttl 1]
15:00:44.658246 rule 20/(ip-option) pass in on em2: 10.20.1.2 >
239.255.255.250: igmp nreport 239.255.255.250 [ttl 1]
15:00:44.654377 rule 20/(ip-option) pass in on em2: 10.20.0.1 >
239.255.255.250: igmp nreport 239.255.255.250 [ttl 1]
15:00:44.654387 rule 20/(ip-option) pass in on em2: 10.20.0.1 >
239.255.255.250: igmp nreport 239.255.255.250 [ttl 1]
15:00:44.658237 rule 20/(ip-option) pass in on em2: 10.20.1.2 >
239.255.255.250: igmp nreport 239.255.255.250 [ttl 1]
15:00:44.658246 rule 20/(ip-option) pass in on em2: 10.20.1.2 >
239.255.255.250: igmp nreport 239.255.255.250 [ttl 1]
15:00:44.835776 rule 0/(match) block in on em0: 121.32.2.184.27436 >
192.75.14.246.56294: udp 98
```

## Appendix D – Firewall Filter Script Output 1

Note 1: The most important code line in the script to get the output 1(with port numbers):

```
#pattern = re.compile('\s(block).*:\s(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,5})\.*')
```

Note 2: Explanations of the log file output:

Total of X scrip: The X stands for the sum of each distinct source IP address.

Y: Source IP: The Y stands for the sum of that particular source IP address.

Note 3: It is easy to figure out that the sum, which is quite a big number, of Y would be the total number of entries that pf captured.

```
----------------------------------
List of pfLog SrcIP Activities:
----------------------------------
block
                                    Total of 41645 scrip (non-repeats);
7226 : 61.139.219.200.80
4454 : 61.139.219.214.80
4396 : 61.139.219.215.80
2142 : 24.47.153.27.6654
1942 : 123.197.64.75.11205
1765 : 123.112.1.196.7129
1753 : 24.34.56.192.5156
1123 : 222.71.163.66.7124
1097 : 60.50.2.101.4679
1088 : 61.51.202.228.5150
1069 : 222.88.99.194.7566
```

## Appendix F – Firewall Filter Script Output 3

Note: The most important code line in the script to get the output 2(with port numbers):

```
# pattern = re.compile('\s(block).*:\s(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.*')
```

```
----------------------------------
List of pfLog SrcIP Activities:
----------------------------------
block
                                        Total of 23004 scrip(non-repeats);
8726 : 192.75.14.245
7226 : 61.139.219.200
4454 : 61.139.219.214
4396 : 61.139.219.215
2142 : 24.47.153.27
1942 : 123.197.64.75
1765 : 123.112.1.196
1753 : 24.34.56.192
1617 : 122.193.12.8
1305 : 219.147.212.186
1143 : 122.225.55.40
1123 : 222.71.163.66
1097 : 60.50.2.101
1094 : 93.86.17.147
1088 : 61.51.202.228
1069 : 222.88.99.194
1037 : 174.49.74.140
974 : 118.180.237.55
818 : 58.214.0.46
```

## Appendix G – Firewall Filter Script Output 4

Note: The most important code line in the script to get the output 2(with port numbers):

```
# pattern = re.compile('\s(pass).*:\s(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.*')
```

```
----------------------------------
List of pfLog SrcIP Activities:
----------------------------------
pass
                                    Total of 269 srcip (non-repeats);
2134 : 10.20.0.1
1446 : 10.2.2.33
982 : 10.20.5.2
836 : 192.168.0.7
822 : 10.1.2.234
810 : 10.3.1.13
772 : 10.1.1.81
766 : 10.1.2.225
764 : 10.20.0.8
764 : 10.20.0.7
758 : 10.1.2.124
754 : 10.1.1.82
748 : 10.3.1.18
720 : 10.2.3.98
668 : 10.20.1.2
439 : 211.157.102.228
352 : 10.1.2.165
308 : 10.1.3.22
234 : 10.1.2.183
226 : 10.1.2.106
192 : 192.168.0.2
```