ANOMALY DETECTION USING NETWORK TRAFFIC CHARACTERIZATION

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Software

by
Oğuz YARIMTEPE

July 2009
İZMİR

We approve the thesis of **Oğuz YARIMTEPE**

Asst. Prof. Dr. Tuğkan TUĞLULAR
Supervisor

Asst. Prof. Dr. Tuğkan TUĞLULAR
Committee Member

Asst. Prof. Dr. Tolga AYAV
Committee Member

Prof. Dr. Şaban EREN
Committee Member

6 July 2009

Prof. Dr. Sıtkı AYTAÇ
Head of the Computer Engineering Department

Prof. Dr. Hasan BÖKE
Dean of the Graduate School of Engineering and Sciences

ABSTRACT

ANOMALY DETECTION USING NETWORK TRAFFIC CHARACTERIZATION

Detecting suspicious traffic and anomaly sources are a general tendency about approaching the traffic analyzing. Since the necessity of detecting anomalies, different approaches are developed with their software candidates. Either event based or signature based anomaly detection mechanism can be applied to analyze network traffic. Signature based approaches require the detected signatures of the past anomalies though event based approaches propose a more flexible approach that is defining application level abnormal anomalies is possible. Both approach focus on the implementing and defining abnormal traffic. The problem about anomaly is that there is not a common definition of anomaly for all protocols or malicious attacks. In this thesis it is aimed to define the non-malicious traffic and extract it, so that the rest is marked as suspicious traffic for further traffic. To achieve this approach, a method and its software application to identify IP sessions, based on statistical metrics of the packet flows are presented. An adaptive network flow knowledge-base is derived. The knowledge-base is constructed using calculated flows attributes. A method to define known traffic is displayed by using the derived flow attributes. By using the attributes, analyzed flow is categorized as a known application level protocol. It is also explained a mathematical model to analyze the undefined traffic to display network traffic anomalies. The mathematical model is based on principle component analysis which is applied on the origin-destination pair flows. By using metric based traffic characterization and principle component analysis it is observed that network traffic can be analyzed and some anomalies can be detected.

ÖZET

AĞ TRAFİĞİ KARAKTERİSTİĞİNİ KULLANARAK ANOMALİ TESPİTİ

Trafik analizindeki en temel yaklaşımlardan birisi de şüpheli trafiğin tespit edilmesidir. Network trafiği ile ilgili anomali tespitine olan ihtiyaçtan dolayı farklı yaklaşımlar ve bunların yazılım çözümleri geliştirilmiştir. Network trafiğinin incelenmesinde olay tabanlı veya imza tabanlı bir yaklaşım sergilenebilir. İmza tabanlı yaklaşımlar önceden yaşanmış anormalliklerden çıkarılan imzalara dayanırken olay tabanlı yaklaşımlar daha esnek bir şekilde anormalliklerin ifade edilebilmesini sağlar. Her iki yaklaşımda da anormal trafiğin ifade edilebilmesi gerekmektedir. Anomali ile ilgili genel sorun ise, her protokol ve durum için genel bir ifade biçimin olmayışıdır. Bu tez çalışmasında, normal trafiğin tanımlanması amaçlanmıştır. Gözlemlenen trafikten normal olarak tanımlanan trafik çıkarılarak kalan trafiğin şüpheli olarak incelenmesi hedeflenmiştir. Bu hedefi gerçeklemek için IP oturumlarına ve istatistiksel metrik değerlerine bağlı ağ paket akışları kullanılmıştır. Ağ akışları ile ilgili gerçeklenebilir ve ağdaki akışların davranış özelliklerini ifade eden bir veri tabanı oluşturulmuştur. Akış özelliklerinden yola çıkarak trafik karakteristiği çıkarma yöntemi açıklanmıştır. Akış özellik değerleri kullanılarak trafik karakteristiğinin nasıl yapıldığı gösterilmiştir. Ayrıca, ele alınan trafik ile ilgili anormallik tespitinde kullanılabilmesi için de matematiksel bir model açıklanmıştır. Birincil Bileşen Analizi (Principle Component Analysis) isimli bu yöntem ile kaynak-hedef çiftlerini içeren akışlar için grafiksel olarak anomali tespit edilebildiği gösterilmiştir. Böylece, incelenen trafiğin karakteristiği çıkarılarak şüpheli trafik üzerinde nasıl anomali tespiti yapılacağı açıklanmıştır.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Network traffic measurement provides basic traffic characteristics, supplies information for the control of the network, allows modeling and provides an opportunity to develop and plan the use of network resources. It also enables developers to control the quality of network service operations. Although network traffic measurement is a well-known and applicable area, a general method for detecting anomalies in network traffic is an important, unsolved problem (Denning 1986).

Anomaly detection can be described as an alarm for strange system behavior. The concept stems from a paper fundamental to the field of security, An Intrusion Detection Model, by *Denning* (Denning 1986). In it, she describes building an "activity profile" of normal usage over an interval of time. Once in place, the profile is compared against real time events. Anything that deviates from the baseline, or the norm, is logged as anomalous. So, anomaly detection systems establish a baseline of normal usage patterns, and anything that widely deviates from it gets flagged as a possible intrusion. A good example for this approach is Bro-IDS (Bro-IDS 2009). It works as en event based intrusion detection system, that is it does not rely on only signatures, recorded and generated by observing previously seen anomalies, but also event definitions that enables dynamic approach to intrusion detection systems. By using its own language, it is possible to define anomalies on application level. Such an event based approach enable further protection for the unseen anomalies. Another anomaly detection system works using signatures that are previously recorded. Since its dependency to previous anomalies and signature collection, signature based systems are not as dynamic as event based systems.

The network traffic to be an anomaly can vary, but normally, any incident that occurs on frequency greater than or less than two standard deviations from the statistical norm can be approached as a suspicious event. Since the ambiguity on determination of the statistical norm, a general method for detecting anomalies in network traffic doesn't have a unique solution. Basically, it should be possible to observe most anomaly types by inspecting traffic flows. However, to date, there is not a common approach to anomaly detection. There are many good reasons for this: Traffic flows present many possible types of network traffic, the set of all flows occupies a very high-dimensional space, and collecting all traffic flows is very resource-intensive.

In this thesis, detecting anomalies are achieved neither signature nor signature based. Both approaches require definition of an anomaly either in a signature way or an application level protocol way. Instead of defining abnormal traffic, it is presented that, defining normal traffic behavior is easier. By using the normal traffic characteristics, the rest of the traffic can be extracted as suspicious traffic for anomaly detection.

Throughout this thesis, it is shown that traffic flow attributes can be used to define known traffic, which covers application level protocols like FTP, TELNET, SSH, HTTP, HTTPS, IMAP, POP, SMTP, MSN CHAT, RLogin, BitTorrent. Network traffic is taken into consideration as combination of directional flows. For each flow, flow attributes are calculated either using flow metrics or packet based inspection. Packet based inspection includes traversing through the collected packets that belong to a flow session. Especially, attribute values related with statistical analysis requires packet based inspection.

By using the attribute values, it is possible to calculate a match result for each application level protocol. By looking at the match result of the calculated values, it is seen that it is possible to define a threshold value for each protocol and any flow that is under the defined threshold value can be marked as undefined for further inspection. When the undefined flows are observed as a whole and principle component analysis is applied over byte and packet number level, it is seen that the generated graphs has peeks that displays suspicious anomalies over time intervals. For calculating the principle components, number of packets, number of bytes and number of IP flow values are used for each origin-destination pairs. Origin-destination pairs are calculated for the splitted time series of the undefined traffic.

The exact number of applications that contribute to the network traffic is not known and even the actual impact of well-known protocols is not clear. This is due to the shortcomings of the state-of-the-art in traffic monitoring, which make use of registered and well-known port numbers to classify packets and computer statistics. The problem is new applications often do not use a registered port, do not have a fixed port number, or simply disguise themselves using the port numbers of another application (for example, the web's port 80) to avoid detection (so they can pass through firewalls, and avoid rate limits) (Hernandez-Campos, et. al. 2005).

This thesis will cover mathematical models for traffic characterization, anomaly detection and implementation of them on network traffic. Chapter 2 aims to give a background information. Flow explanations like directionality or flow metrics and tools that are used either on packet or flow based are explained throughout this chapter. This chapter also covers the background information about principle component analysis that is used for anomaly detection for the suspicious traffic. Chapter 3 explains the model of the solution. It is aimed to answer the *how* part of the thesis. Before starting the implementation phase it is aimed to give logical methodical steps that will be follow to gather anomalies at the network traffic. Chapter 4 covers the implementation details starting from the programming language itself to UML diagrams of the class hierarchy. Chapter 5 includes test results of the particular application level protocols. The results are used to detect threshold for each protocol. Chapter 5 also includes the test results related with the BitTorrent traffic which is added to this thesis as a contribution for the related work. The thesis ends with a conclusion and a further work part.

CHAPTER 2

BACKGROUND

This section aims to provide a background for the concepts used in this thesis. To provide the reader a familiarity with the topics covered through this thesis, this section is divided into three subsections. First section is dedicated for network flow explanations. Flow notion, flow based metrics, directionality and flow attributes are explained in detail. This category also includes the tools used for calculating flow metrics. The second sub section is for packet based inspection and packet based tools used throughout this thesis. In this thesis work, it is also mentioned an anomaly detection method that can be applied on to the flow records, which is called principle component analysis (PCA). Mathematical details about PCA calculation is given at the third subsection of this section.

2.1. Network Flows

2.1.1. Notion of Flow

The notion of flow was introduced within the network research community in order to better understand the nature of Internet traffic. Flow is the sequence of packets or a packet that belonged to certain network session(conversation) between two hosts but delimited by the setting of flow generation or analyzing tool (Lee 2008, June). Network flow data represents a summary of conversation between two end points. It provides valuable information to assist investigation and analysis of network and security issues. Unlike deep packet inspection, flow data does not rely on packet payloads. Instead the analyst relies on information gathered from packet headers and its associated metrics. This provides the analyst a neutral view of network traffic flow by tracking network sessions between multiple endpoints simultaneously. In addition, having network flow data will provide a better visibility of network events without having the need to perform payload analysis. It is convenient for protocol analysis (Lee 2008, May) or debugging.

In 1995, the IETF's Realtime Traffic Flow Measurement (RTFM) working group was formed to develop a frame-work for real-time traffic data reduction and measurements (RTFM 1995). A flow in the RTFM model can be loosely defined as the set of packets that have in common values of certain fields found in headers of packets. The fields used to aggregate traffic typically specify addresses at various levels of the protocol stack (e.g. IP addresses, IP protocol, TCP/UDP port numbers). Herein, it is used the term key to refer to the set of address attributes used to aggregate packets into a flow.

2.1.2. Flow Directionality

Flow definition is given above as summary of conversation between two end points. The endpoints here are defined as follows:

a. Layer 2 Endpoint - Source Mac Address | Destination Mac Address
Layer 3 Endpoint - Source IP Address | Destination IP Address
Layer 4 Endpoint - Source Port | Destination Port

The conversation between these two ends has a direction so flow tool display a direction information related with the flow also. There are two types of direction information related with a network flow. A flow can be defined either as a unidirectional flow or as a bidirectional flow.

At unidirectional flow model, every flow record contains the attribute of single endpoint only. Figure 2.1 and Figure 2.2 show directionality in a more simple way (Lee

2008, June).

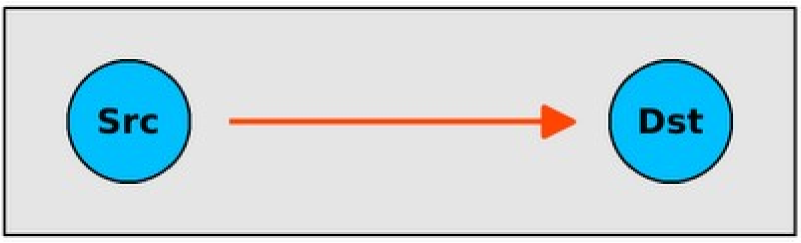


Figure 2.1. Unidirectional Flow Demonstration, From Source to Destination

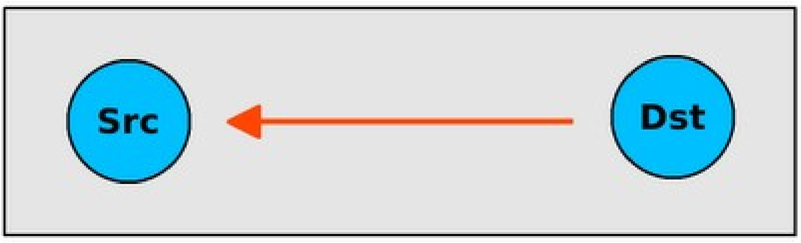


Figure 2.2. Unidirectional Flow Demonstration, From Destination to Source

At bidirectional flow model every flow record contains the attribute of both endpoints. Figure 2.2 illustrates it (Lee 2008, June):

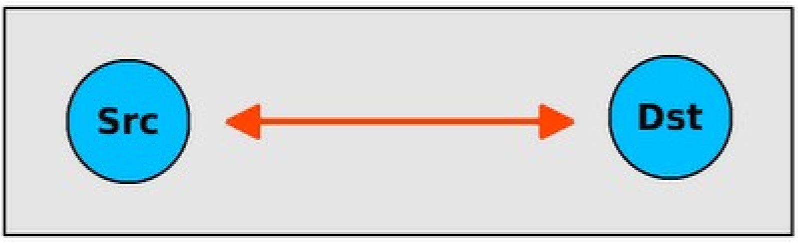


Figure 2.3. Bidirectional Flow Between Two Endpoints

To make the directionality more clear lets assume that source host sends 90 bytes to destination host and destination host replies with 120 bytes. If the flow communication between these two hosts are unidirectional, then the flow information will be as follows:

b.

Srcaddr	Direction	Dstaddr	Total Bytes
Source Host	->	Destination Host	90
Destination Host	->	Source Host	120

Though, if the flow communication is bidirectional, then the result will be different:

c.

Srcaddr	Direction	Dstaddr	Total Bytes	Src Bytes	Dst Bytes
Source Host	<->	Destination Host	210	90	120

In unidirectional flow, it is only seen the total bytes that sent by source host but nothing about destination in the first flow record. Then the next record shows destination sends 120 bytes to source. The total bytes is accounted from single endpoint only. But in bidirectional flow, it can be seen that source host sends 90 bytes and destination replies with 120 bytes. The total bytes is the accumulation of source and destination bytes.

2.1.3. Flow Identification

Giving a unique label to a flow information depends on the aim of the analysis related with it. If the intent is to analyze the amount of traffic between two hosts, then the focus can be source and destination IP addresses. However, a finer intent like considering the flow information over a state approach to identify connections will require additional information. To identify each flow it is used 5-tuple key representation. The tuple includes source IP address, source port number, destination IP address, destination port number and protocol information.

2.1.4. Flow Period

Flow period means the time periodically report on a flow's activity. The period determines the start and end times of each flow. There are three primary expiry methods that are appropriate for studying characteristics of individual flows: protocol based, fixed timeout, and adaptive timeout (Keys, et. al. 2001). With protocol based mechanisms, the state of a flow is determined by observing protocol specific messages (e.g. TCP SYN, FIN or RST). With a fixed timeout method, a flow has expired when some fixed period has elapsed since the last packet belonging to that flow was captured. An adaptive timeout strategy is a little more sophisticated than a fixed timeout method. The timeout is different for each flow and is computed based on the packet rate observed so far within each flow.

In this thesis, because of its simplicity a fixed timeout approach is chosen over an adaptive timeout mechanism to decide the expiration time of flows. The timeout value is chosen as 60 seconds for preliminary examination, which is also commonly used in related works (Xu, et.al. 2005) (Claffy, et.al. 1995) (Karagiannis, et.al. 2005).

2.1.5. Flow Attributes

In the literature, flow attributes are often called features, or characteristics. According to the RFTM Architecture (RFC2722 1999) a flow has computed attributes that are derived from end point attribute values, metric values like packet and byte counts, time values as well as summary information like mean, median or average values, jitters and distributional information. The goal in defining flow attributes is to identify now only the relevant characteristics but also the proper way to measure them.

In 2005, *De Montigny* and *Leboeuf* published the definition of flow attributes that can be used to characterize a flow at their paper (De Montigny and Leboeuf 2005). At their work it is mentioned nearly thirty property that can be derived from a flow information. The mentioned flow metrics are also used throughout this thesis. The following sub sections will be covering the details of the flow attributes mentioned at the *De Montigny* and *Leboeuf's* work (De Montigny and Leboeuf 2005).

Flow attributes are examined in two categories. One of the categories includes the whole flow values, while the second one includes values per directional flows. The attributes derived are summarized in Table 2.1 and Table 2.2. The following sub sections will describe each attribute with greater detail. Table 2.1 lists the attributes that are measured over the entire flow. The first three attributes (Key, BeginTime, EndTime) are simply used to identify and sort the flows. Table 2.2 gives the attributes that are specific to each direction, and thus are measured in each direction separately. The details of Table 2.1 and Table 2.2 can be found at Appendix A.

Table 2.1. Attributes measured over the whole flow

Attributes	**Inspection Method**
KEY	Flow Based
BEGIN_TIME	Flow Based
END_TIME	Flow Based
DURATION	Flow Based
FIRST_NONEMPTY_PACKET_SIZE	Packet Based
FIRST_FEW_NONEMPTY_PACKET_DIRECTIONS	Packet Based
DATA_BYTE_RATIO_ORIG_TO_RESP	Flow Based
INTERARRIVAL_DISTRIBUTION	Packet Based
Conversational Indicator	
$\text{ALPHA}_{\text{conversation}}$	Packet Based
$\text{BETA}_{\text{conversation}}$	Packet Based
$\text{GAMMA}_{\text{conversation}}$	Packet Based
Transaction Indicator	
$\text{ALPHA}_{\text{transaction}}$	Packet Based

Table 2.2. Attributes measured for each direction of the flow

Attributes	**Inspection Method**
INTERARRIVAL_DISTRIBUTION	Packet Based
PAYLOAD_DISTRIBUTION	Packet Based
BYTE_COUNT	Flow Based
DATA_BYTE_COUNT	Flow Based
PACKET_COUNT	Flow Based
DATAPACKET_COUNT	Flow Based
Encryption Indicators	
$\text{ALPHA}_{\text{chipherblock}}$	Packet Based
$\text{BETA}_{\text{chipherblock}}$	Packet Based
Keystroke Interactive Indicator	
$\text{ALPHA}_{\text{key\_interactive}}$	Packet Based
$\text{BETA}_{\text{key\_interactive}}$	Packet Based
$\text{GAMMA}_{\text{key\_interactive}}$	Packet Based
$\text{DELTA}_{\text{key\_interactive}}$	Packet Based
$\text{EPSILON}_{\text{key\_interactive}}$	Packet Based
Command-line Interactive Indicator	
$\text{ALPHA}_{\text{cmd\_interactive}}$	Packet Based
$\text{BETA}_{\text{cmd\_interactive}}$	Packet Based
$\text{GAMMA}_{\text{cmd\_interactive}}$	Packet Based
$\text{DELTA}_{\text{cmd\_interactive}}$	Packet Based
$\text{EPSILON}_{\text{cmd\_interactive}}$	Packet Based
File transfer Indicators	
$\text{ALPHA}_{\text{constantpacketrate}}$	Packet Based
$\text{BETA}_{\text{file}}$	Packet Based
$\text{GAMMA}_{\textbf{file}}$	Packet Based

As it can be seen from Table 2.1 and Table 2.2, attributes related with whole flow or each direction of flow have indicators. Indicators generally depend on packet based calculations and enables to derive application level information from flow itself. Although it is mentioned they are calculated mainly by packet based inspection, some of them also use flow based data to calculate some indicator values. Details about the indicators are mentioned at the next section.

2.2. Network Flow Tools

Custom flow tools ease the work on flow data. They reconstructs the actual data streams and enable it to be saved to a file in a formated way or to be displayed graphically. What they do basically is understanding sequence numbers and state information to decide to which session of the packages belongs to.

Cisco's NetFlow (Cisco Systems 2009) is a network protocol developed by Cisco Systems to run on Cisco IOS-enabled equipment for collecting IP traffic information. It's proprietary and supported by platforms other than IOS, such as Juniper routers or FreeBSD and OpenBSD (Netflow 2009). Although it is widely used, its flows are unidirectional and limited number of flow attributes are recorded. NetFlow traffic is mainly analyzed by adapting other tools to the network like cflowd (CAIDA 2006) and SiLK (SiLK, 2009). Another popular tool is Argus, which is fixed-model real time flow monitor designed to track and report on the status and performance of all network transactions seen in a data network traffic stream.

2.2.1 Argus

The Argus Open Project is focused on developing network audit strategies that can do real work for the network architect, administrator and network user. Argus is a fixed-model real time flow monitor designed to track and report on the status and performance of all network transactions seen in a data network traffic stream. Argus provides a common data format for reporting flow metrics such as connectivity, capacity, demand, loss, delay, and jitter on a per transaction basis. The record format that Argus uses is flexible and extensible, supporting generic flow identifiers and metrics, as well as application/protocol specific information.

Argus can be used to analyze and report on the contents of packet capture files or it can run as a continuous monitor, examining data from a live interface, generating an audit log of all the network activity seen in the packet stream. Argus currently runs on Linux, Solaris, FreeBSD, OpenBSD, NetBSD, MAC OS X and OpenWrt (ARGUS 2009).

Argus is used for converting previously recorded pcap files to its own flows format and to produce meaningful human readable flow information from it. It is used as follows:

d.

```
argus  -mAJZR  -r ettercap-dos.pcap  -w ettercap-dos.pcap.arg3
```

Here, ettercap-dos.pcap file is converted into an Argus flow format by using the parameters defined below:

e.

```
-m:  Provide MAC addresses information in argus records.
-A: Generate application byte metrics in each audit record.
-J: Generate packet peformance data in each audit record.
-Z: Generate packet size data.
-R: Generate argus records such that response times can be derived from
transaction data.
```

Converted Argus flow file is a binary file which requires Argus client tools to be used for meaningful information. Racluster is one of the Argus client tools that is used for gathering flow information. Racluster reads Argus data from an Argus data source, and clusters/merges the records based on the flow key criteria specified either on the command line, or in a racluster configuration file, and outputs a valid Argus stream. This tool is primarily used for data mining, data management and report generation. Below is a sample usage and the produced output of it:

f.

```
racluster -L0 -nr ettercap-dos.pcap.arg3 -s proto saddr sport dir daddr dport

Proto          SrcAddr  Sport  Dir          DstAddr  Dport
tcp     192.168.1.118.32743    ->     192.168.1.188.2
tcp     192.168.1.118.32999    ->     192.168.1.188.3
```

By looking at the produced output, it can be said that, the first flow indicates a connection over TCP between the IP addresses 192.168.1.118 and 192.168.1.188 that is also a unidirectional flow. During the flow inspection process more information rather than the ones mentioned above is gathered using racluster, like the start time of flow, duration, number of packets send by source, ... etc.

Another Argus client was rasplit which was used for splitting flow sources into sub flows depending on either time or size values. Rasplit reads Argus data from an Argus data source, and splits the resulting output into consecutive sections of records based on size, count time, or flow event, writing the output into a set of output files. This tool is mainly used at the principle component analysis phase for creating sub flows.

2.3. Network Packets

In information technology, a packet is a formatted unit of data carried by a packet mode computer network. Computer communications links that do not support packets, such as traditional point-to-point telecommunications links, simply transmit data as a series of bytes, characters, or bits alone. When data is formatted into packets, the bitrate of the communication medium can better be shared among users than if the network were circuit switched (Packet 2009).

The format of the network packets are defined as protocols. Throughout this thesis, User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) packets are taken into consideration because of their common usage at application level. So is this thesis, packet is used as either a TCP or an UDP packet.

Each protocol has its own header definitions. Headers carry details about network packets. For both UDP and TCP, it is gathered common header information for each packet. One of the gathered header information is the IP addresses, that are defined in 32 bit fields in dot separated format for IPv4. It should be mentioned that, current work in this thesis is done on IPv4 networks. An IP packet contains two IP address information that is the sender/source IP address and the other one is the receiver/destination IP address. Packets are left the machines or received by using port numbers which are defined as 8 bit information at the packet headers. An IP packets also carry protocol information, which is defined in 8 bit fields. This number is 17 for TCP protocol packets and 6 for UDP packets when converted into decimal value.

Each packet has a length information that is defined in a 24 bit header length.

This gives the total length of the packet. The most important header part for this thesis is the payload. Payloads are the data carriage of the packets. Protocol related commands, text information or binary data is carried on payload parts. Depending on how the network traffic is sniffed and the length of the captured packets defined, the size of payload can vary. But in general, the size and ingredients of it gives much information related with which application the packet belongs to.

2.4. Network Packet Tools

This section is covering the tools that gathers packet level data from network traffic. These tools generally use libpcap (Libpcap 2009) library for low-level network jobs.

2.4.1. Tcpdump

Tcpdump is a common packet sniffer that runs under the command line. It allows the user to intercept and display TCP/IP and other packets being transmitted or received over a network to which the computer is attached (Tcpdump 2009). It prints out a description of the contents of packets on a network interface that match the boolean expression. It can also be run with the -w flag, which causes it to save the packet data to a file for later analysis, and/or with the -r flag, which causes it to read from a saved packet file rather than to read packets from a network interface. In all cases, only packets that match expression will be processed by tcpdump.

Tcpdump is used in this thesis to save the manually produced attacks as pcap format for further investigation. The saved files are processed via the developed software and a general anomaly graph is produced for an attack sample.

2.4.2. Wireshark

Wireshark is a free packet sniffer computer application. It is used for network troubleshooting, analysis, software and communications protocol development, and education.

Wireshark is very similar to tcpdump, but it has a graphical front-end, and many more information sorting and filtering options. It allows the user to see all traffic being passed over the network (usually an Ethernet network but support is being added for others) by putting the network interface into promiscuous mode (Wireshark 2009).

It is used as a controller through the development of flow attributes. By inspecting each packets per flow, it is decided to understand the which header fields should be taken into consideration for each protocol.

2.4.3. Tcpreplay

Tcpreplay is a tool for replaying network traffic from files saved with tcpdump or other tools which write pcap files. It allows one to classify traffic as client or server, rewrite Layer 2, 3 and 4 headers and finally replay the traffic back onto the network and through other devices such as switches, routers, firewalls, NIDS and IPS's. Tcpreplay supports both single and dual NIC modes for testing both sniffing and inline devices (TcpReplay 2009).

Throughout this thesis, it is used to reproduce the captured manual attack over an Ethernet interface to be sniffed and analyzed by Snort. It is aimed to check the produced anomaly results with the Snort results, so the undefined flows are sent to Snort after being analyzed.

2.5. Principle Component Analysis

Principal component analysis (PCA) involves a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components (PCA 2009). With minimal effort PCA provides a road map for how to reduce a complex data set to a lower dimension to reveal the sometimes hidden, simplified structures that often underlie it.

Principal component analysis is based on the statistical representation of a random variable. Suppose we have a random vector population x, where

$$x=(x_1,x_2,\ldots,x_n)^T \tag{2.1}$$

and the mean of that population is denoted by

$$\mu_x=E\{x\} \tag{2.2}$$

and the covariance matrix of the same data set is

$$C_x=E\{(x-\mu_x)(x-\mu_x)^T\} \tag{2.3}$$

The components of C_x, denoted by c_{ij}, represent the covariances between the random variable components x_i, and x_j. The component c_{ii} is the variance of the component x_i. The variance of a component indicates the spread of the component values around its mean value. If two components x_i and x_j of the data are uncorrelated, their covariance is zero ($c_{ij} = c_{ji} = 0$). The covariance matrix is, by definition, always is always zero.

From a sample of vectors $x_1,\ldots,x_M$ it can be calculated the sample mean and the sample covariance matrix as the estimates of the mean and the covariance matrix. From a symmetric matrix such as the covariance matrix, it can be calculated an orthogonal basis by finding its eigenvalues and eigenvectors. The eigenvectors e_i The eigenvectors

λ_i are the solutions of the equation.

$$C_x e_i = \lambda_i e_i,\ i=1,...,n \tag{2.4}$$

For simplicity it is assumed that the λ_i are distinct. These values can be found, for example, by finding the solutions of the characteristic equation

$$|C_x - \lambda I| = 0 \tag{2.5}$$

where the I is the identity matrix having the same order than C_x and the |.| denotes the determinant of the matrix. If the data vector has n components, the characteristic equation becomes of order n. This is easy to solve only if n is small. Solving eigenvalues and corresponding eigenvectors is a non-trivial task, and many methods exist.

By ordering the eigenvectors in the order of descending eigenvalues (largest first), one can create an ordered orthogonal basis with the first eigenvector having the direction of largest variance of the data. In this way, it can be found directions in which the data set has the most significant amounts of energy.

Suppose one has a data set of which the sample mean and the covariance matrix have been calculated. Let A be a matrix consisting of eigenvectors of the covariance matrix as the row vectors. By transforming a data vector x, it is got

$$y = A(x-\mu_x) \tag{2.7}$$

which is a point in the orthogonal coordinate system defined by the eigenvectors. Components of y can be seen as the coordinates in the orthogonal base. It can be reconstructed the original data vector x from y by

$$x = A^T y + \mu_x \tag{2.8}$$

using the property of an orthogonal matrix $A^{-1} = A^T$. The A^T is the transpose of a matrix of A. The original vector x was projected on the coordinate axes defined by the orthogonal basis. The original vector was then reconstructed by a linear combination of the orthogonal basis vectors.

Instead of using all the eigenvectors of the covariance matrix, the data is represented in terms of only a few basis vectors of the orthogonal basis. If the matrixis denoted having the K first eigenvectors as rows by A_K a similar transformation can be created as seen above

$$y = A_K (x - \mu_x) \tag{2.9}$$

and

$$x = A_K^T y + \mu_x \tag{2.10}$$

This means that it is projected the original data vector on the coordinate axes having the dimension K and transforming the vector back by a linear combination of the basis vectors. This minimizes the mean-square error between the data and this representation with given number of eigenvectors.

If the data is concentrated in a linear subspace, this provides a way to compress data without losing much information and simplifying the representation. By picking the eigenvectors having the largest eigenvalues it is lost as little information as possible in the mean-square sense. One can e.g. choose a fixed number of eigenvectors and their respective eigenvalues and get a consistent representation, or abstraction of the data. This preserves a varying amount of energy of the original data. Alternatively, it can be chosen approximately the same amount of energy and a varying amount of eigenvectors and their respective eigenvalues. This would in turn give approximately consistent amount of information in the expense of varying representations with regard to the dimension of the subspace (Hollmen 1996).

2.5.1. Tools Used Through PCA Process

PCA process is a manual process. This section describes the details about the tools used throughout the PCA steps.

2.5.1.1. Octave

GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab. It may also be used as a batch-oriented language[13].

It is used for principle component calculation and generating graphs of the calculation to see the peeks at the graphs.

GNU Octave's statistical package has *princomp* function which enables computing the PCA of a X matrix. It works as follows:

g.

```
[pc,z,w,Tsq] = princomp(X)
pc:  the principal components
z :  the transformed data
w:   the eigenvalues of the covariance matrix
Tsq: Hotelling's T^2 statistic for the transformed data
```

Throughout PCA process, anomaly detection is done over undefined flow records. While Octave is used for PCA computation and generating anomaly graphs, some other tools are used for creating manual attacks and retesting them.

2.5.1.2. Snort

SNORT is an open source network intrusion prevention and detection system utilizing a rule-driven language, which combines the benefits of signature, protocol and anomaly based inspection methods. Snort is the most widely deployed intrusion detection and prevention technology worldwide and has become the de facto standard for the industry (Snort 2009).

Snort is used in intrusion detection mode to detect the anomalies for the undefined traffic. Snort BASE (BASE 2009) web interface is used for observing the produced results.

2.5.1.3. Ettercap

Ettercap is a suite for man in the middle attacks on LAN. It features sniffing of live connections, content filtering on the fly and many other interesting tricks. It supports active and passive dissection of many protocols (even ciphered ones) and includes many feature for network and host analysis (Ettercap 2009).

Ettercap is able to perform attacks against the ARP protocol by positioning itself as "man in the middle" and, once positioned as this, it is able to:

- infect, replace, delete data in a connection
- discover passwords for protocols such as FTP, HTTP, POP, SSH1, etc ...
- provide fake SSL certificates in HTTPS sections to the victims.

It is used for producing DOS attacks, manually. The produced dos attacks are saved by using tcpdump, and analyzed both with Snort and developed software.

2.5.1.4. Nmap

Nmap ("Network Mapper") is a free and open source (license) utility for network exploration or security auditing. Many systems and network administrators also find it useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. It was designed to rapidly scan large networks, but works fine against single hosts. Nmap runs on all major computer operating systems, and both console and graphical versions are available (Nmap 2009). It is used as a port scanner to generate port scan traffic over a target machine.

CHAPTER 3

APPROACH

This chapter explains the methodical work that is followed to characterize network traffic and to get anomaly information related with the traffic examined. The method involves the steps followed to produce anomaly result. The steps start with examining of the off line prerecorded data and ends with an graph representing the abnormal traffic in a time interval. Each step is achieved by considering some key points which are also mentioned at the following sub sections. The key points constitute a unique view which differs this thesis work from the similar works.

3.1. Principle of Approach

The method that is followed at this work can be viewed at four steps. To get a general view the steps are explained in a simple way. The details are explained at the following subsections.

- <u>Packets are grouped into flows</u>: Off line data is used at this thesis. The details about the captured data is given at the subsection 3.2. The data is a prerecorded data that includes captured traffic by tcpdump. The recorded data is first converted into a flow record by keeping mac address information, packet performance data, application byte metrics in each audit record.
- <u>Characteristics (attributes) are measured on each flow</u>: Attributes mentioned at the Annie De Montigny and Leboeufare (De Montigny and Leboeuf 2005) work are calculated and recorded.
- <u>Flows are recognized and described</u>: The flow is described with its two main

properties. One category includes the metric values of the flow, the other one is the statistical information that is gathered through a packet inspection. With these knowledges it is aimed to define the application level definition for a flow.

- Anomaly Detection: Anomaly analysis is covered on the undefined flow data. Undefined traffic is saved for a further statistical analysis to detect uncorrelated data from the correlated data.

The process is outlined also at Figure 3.1 in a more visual way.

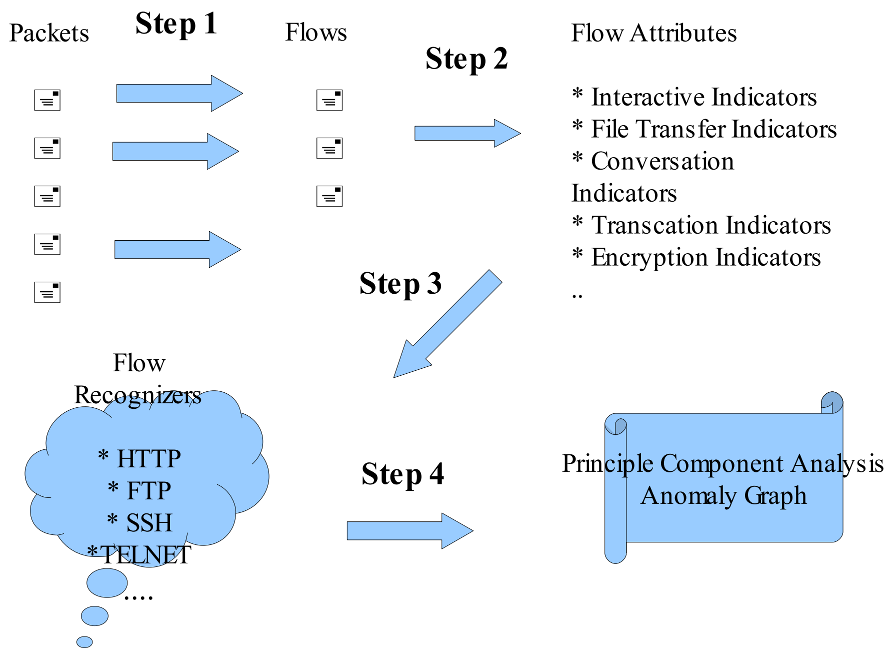


Figure 3.1. Steps Through Anomaly Detection

There are some points that should be highlighted about the approach in this thesis:

- Anomaly detection method at this thesis is based on traffic characterization which requires to derive flow characteristics. At the current work it is possible to get and analyze nearly twenty three flow metrics.
- It is completely avoided relying on port numbers or payload analysis. This provides an alternative method to more conventional traffic categorization techniques provided by current networking tools.

- It is only examined communication patterns found at the network and transport layers, requiring minimal information per packet to be retained.
- Patterns are identified at the 5-tuple flow granularity of TCP/UDP communications. Therefore even sporadic malicious activities may be identified without the requirement of waiting until multiple connections can be examined.
- The flow attributes can serve as a starting point for different traffic characterization studies. By using the same attributes but different statistical analysis more powerful techniques can be defined. The methodology also open to adding to new flow attribute additions. So defining new attributes may ease the anomaly detection process.

3.2. Processing of Captured Data

The traffic analysis process starts with a tcpdump data file which is used for extracting flow data and for gathering per packet information. There are two types of tcpdump records that was used through this thesis work. One of them is the 1999 DARPA Intrusion Detection Evaluation Data Set which was used throughout the development phase. DARPA Intrusion Detection Evaluation Data Set (DARPA 2009) includes weekly prerecorded tcpdump files for further evaluation. The data set was used for intrusion detection so it has separated clean traffic. Attack free (clean) traffic was important during the development period to detect the metric values for each flow attributes. So the first week of the dataset is used for development purposes. Second record type is the one including manually produced anomalies. These records include both abnormal traffic and a scheduled attack produced manually. These type of records are used for checking the accuracy of the work.

Below is the graphical and statistical representation of the week one day one record of the DARPA Intrusion Detection Evaluation Data Set:

h. File name: inside.tcpdump_w1_d1 File type: Wireshark/tcpdump/... - libpcap
File encapsulation: Ethernet
Number of packets: 1492331
File size: 341027537 bytes
Data size: 317150217 bytes
Capture duration: 79210.265570 seconds
Start time: Mon Mar 1 15:00:05 1999
End time: Tue Mar 2 13:00:16 1999
Data rate: 4003.90 bytes/s
Data rate: 32031.22 bits/s
Average packet size: 212.52 bytes

h. represents the all available statistical information gathered from the captured file, inside.tcpdump_w1_d1 which is the name of the file created by using Tcpdump.

Following two figures give the graphical representation of the flow information for the captured file.

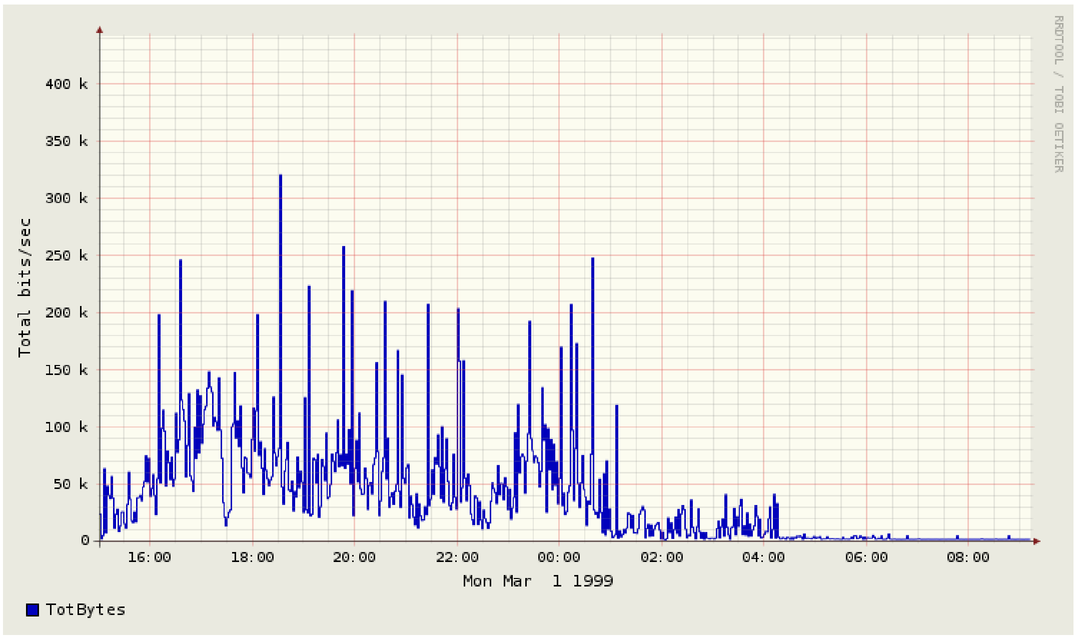


Figure 3.2. Total Bits Per Second for Week 1 Day 1 Record

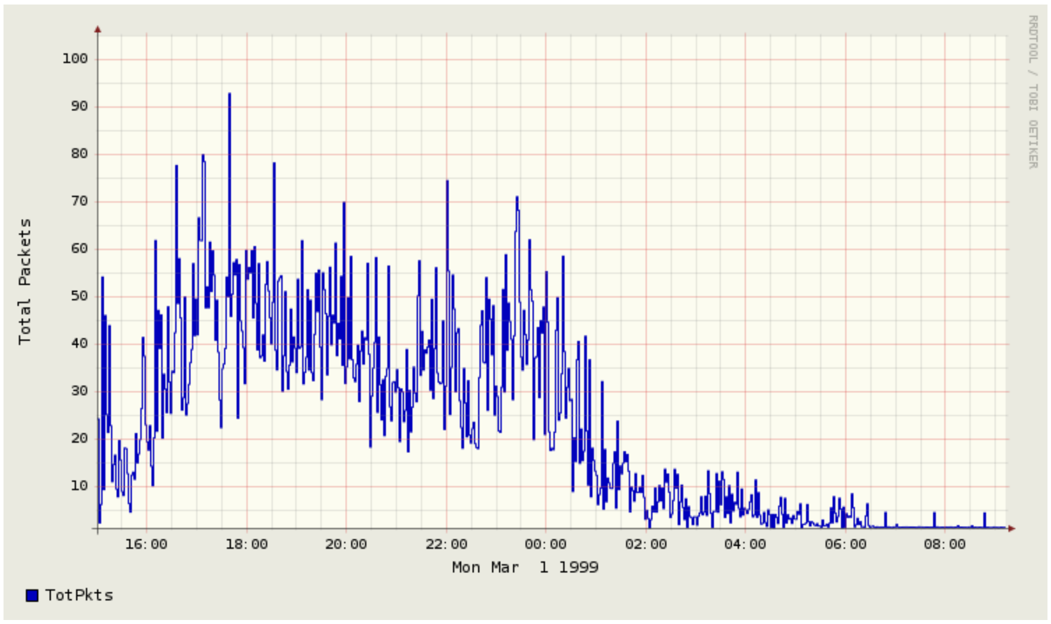


Figure 3.3. Total Number of Packets for Week 1 Day 1 Record

During the preprocessing period, the prerecorded dump file is first converted into Argus flow file. After this process flow records are written to a text file in a human readable form. The records include 5-tuple key information, direction of transaction, record start time, record last time, record total duration, transaction bytes from source to destination, transaction bytes from destination to source, application bytes from source to destination, application bytes from destination to source, packet count from source to destination, packet count from destination to source, source packets retransmitted or dropped and destination packets retransmitted or dropped for each flow.

Preprocessing period also requires packet based traversing on the recorded traffic. While traversing on the recorded data, information per packets are saved. To save the packet based information database table created on MySQL database server is used. MySQL is a relational database management system (RDBMS). Because it is a fast, stable and true multi-user, multi-threaded SQL database server it is used to save packet level information. Below is the structure of the table that is used to keep whole flow packet information.

Table 3.1. Table structure for table FlowPkgTbl

Field	Type	Null	Default
orderno	bigint(20)	Yes	NULL
srcip	varchar(20)	Yes	NULL
dstip	varchar(20)	Yes	NULL
dstport	int(11)	Yes	NULL
sport	int(11)	Yes	NULL
arrivaltime	datetime	Yes	NULL
epochtime	decimal(40,10)	Yes	NULL
proto	varchar(10)	Yes	NULL
payloadsize	int(11)	Yes	NULL
payloadinfo	text	Yes	NULL
packetsize	bigint(20)	Yes	NULL

For each of the flow record, attribute values that are mentioned at the Chapter 2 are calculated. As it is mentioned some attribute metrics require packet based inspection that means packets that belong to the analyzed flow should be taken into consideration. To identify in request packets, 5-tuple key value is used from the flow information. Packets including the 5-tuple key information are queried from the FlowPkgTbl. By traversing on the result set and following the calculations mentioned at the Chapter 2, flow attributes are calculated. Packet based attribute calculation requires additional database tables usage.

3.3. Processing of Flow Data

Attribute values are used to define the flow characteristics. For each of the protocol, observed values of the flow attributes are given at the *De Montigny* and *Leboeuf's* work (De Montigny and Leboeuf 2005). The values are added to the Appendix B. Throughout the development phase, to check the correctness of the values defined at the paper, protocol based traffic should be used, so for every application level protocol that is under consideration, specific traffic is extracted from the recorded traffic record. This is done using the racluster, Argus client. Racluster enables filtering traffic. A sample racluster command for filtering HTTP traffic and creating flow data is as follows:

i. racluster -L0 -nr tcpdump-07-05-2009_23:25:24.dump.arg3 -s proto saddr sport dir daddr dport stime ltime dur sbytes sappbytes dappbytes dbytes spkts dpkts sloss dloss - ip and port 80 > tcpdump-07-05-2009_23:25:24.dump.txt

This command takes tcpdump-07-05-2009_23:25:24.dump.arg3 Argus flow data as input and produces clustered readable flow information by filtering IP based traffic and port 80, that is flows with UDP, TCP or ICMP protocols with destination or source port number 80 is taken into consideration.

Another reason to use the filtered flows is the necessity of calculating threshold value for each protocol. By looking at the flow attribute values and the value ranges defined at the *De Montigny* and *Lebouf's* paper, it is only possible to produce the percentage of the relevance for each protocol. By observing the test results, a threshold value is produced for every protocol. Any match result below that percentage is marked as undefined flow for PCA.

Thresholds are calculated for each application level protocol. By using the clean data set and the filtered application level flows, match results for each protocol is calculated. By looking at the average of the collected results a general tendency at the threshold value for each protocol is calculated. Chapter 4 gives the details about the test results.

3.4. Handling Flow Attributes

Flow attributes are mainly calculated on non-empty packages. A non-empty packet is the one carrying a payload. The reason about dealing with non-empty packets is that packets carry either binary data or protocol specific commands through payload. It is clear that the size of the payload may vary depending on the capture snaplen, but the default value of using the first 60 Bytes are enough to decide about the protocol. Generally the size of the payload is used instead of the text based inspection. Depending on the protocol itself, it is seen that size of the payload can bu used for detecting some information about flow. In addition, calculating the size is easier than inspecting the ingredients of the payload.

3.4.1. Discrete Distribution Attributes for Payload and Inter-Packet Delay

Discrete distributions are used for inter-packet delay and packet payload length. Distribution values are especially important for packet payload length. Application protocol overhead may imply that non-empty packets are always greater or equal in length to a given minimum (due to header length). Application negotiation mechanisms may also exhibit a high frequency of packets of special sizes. Moreover, certain applications may have a preferential packet size, and completely avoid sending packets of lengths within a given range. For instance, as noted in (Hernandez-Campos, et. al. 2005), the HTTP-protocol is characterized by many short and long packets. Such characteristics are not effectively reflected by means and variances which may be sensitive to outliers. Discrete distribution attributes are therefore preferred in this work (De Montigny and Leboeuf 2005).

The payload and inter-packet delay delimiters are used as defined at the *De Montigny* and *Leboeuf's* work (De Montigny and Leboeuf 2005).

It is currently used the following bin delimiters for payload length (in bytes):

j. [0-1[, [1-2[, [2-3[, [3-5[, [5-10[, [10-20[, [20-40[, [40-50[, [50-100[, [100-180[, [180-236[, [236-269[, [269-350[, [350-450[, [450-516[, [516-549[, [549-650[, [650-1000[, [1000-1380[, [1380-1381[, [1381-1432[, [1432-1473[, [1473-inf[.

The inter-packet delays are distributed according to bins ranging from 0 to 64 second-delays. It is currently used the following bin delimiters for inter-packet delay (in seconds):

k. [0-0.000001[, [0.000001-0.0001[, [0.0001-0.001[, [0.001-0.01[, [0.01-0.1[, [0.1-1.0[, [1.0-10.0[, [10.0-64.0[, [64.0-inf[.

Below tables display a conceptual illustration.

Table 3.2. Conceptual illustration of discrete payload distribution

Payload	45%			30%	10%	15%			
(bytes)	[0	[1	...	[1000	[1380	[1381	...	[1473	[1500, +]

Table 3.3. Conceptual illustration of discrete packet delay distribution

Inter-packet delay	45%		30%	10%	15%			
(seconds)	[0	[10^{-6}	[0.0001	[0.001	[0.01	[1	[10	[64, +]

According to the Table 2.3, 45% of the packets carried no data, and another 45% of the packets were relatively big, not full size packets but big packets.

3.4.2. Conversations and Transactions

Conversation and transactions are heuristic approaches to the directionality of the whole connection. Packets are treated as a sequence of positive and negative values (the sign of each value indicates the direction of the packet) and the idea is to characterize the changes in sign. Whether they are interactive or machine-driven, applications often exhibit differences with respect to the transaction and conversation indicators.

A conversation episode in this work contains consecutive (back to back) packets in one direction followed by consecutive packets in the other direction. There is also a sustained conversation definition that is the episode containing consecutive packets in one direction, followed by consecutive packets in the opposite, and followed again by consecutive packets in the first direction (e.g. A->B, B->A, A->B).

According to the *De Montigny* and *Leboeuf's* work ALPHA, BETA and GAMMA values of a conversation is calculated as follows (De Montigny and Leboeuf 2005). Let M be the total number of non-empty packets in a flow, let C be the number of non-empty packets associated with a conversation, and ζ be number of non-empty packets associated with a sustained conversation. It is defined $\text{ALPHA}_{\text{conversation}}$ as the number of non-empty packets that belong to a conversation over the total number of non-empty packets:

$$\text{ALPHA}_{\text{conversation}} = \frac{C}{M} \tag{3.1}$$

It is defined $\text{BETA}_{\text{conversation}}$ as the number of non-empty packets that belong to a sustained conversation over the total of non-empty packets that belong to a conversation:

$$\text{BETA}_{\text{conversation}} = \frac{\zeta}{C} \tag{3.2}$$

Let O be the number of non-empty packets associated with a conversation and transmitted by the Originator, it is defined an indicator of symmetry in a conversational flow, $\text{GAMMA}_{\text{conversation}}$ as the proportion of conversation packets that are transmitted by the originator:

$$\text{GAMMA}_{\text{conversation}} = \frac{O}{C} \tag{3.3}$$

A similar approach is applied for the transactions. Transactions include “ping-pong” exchanges, where one packet is followed by a packet coming in the opposite direction. It is quantified this phenomenon by comparing the number of changes in sign effectively seen, with the maximum number of times a change of sign can occur, given the number of positive and negative values in that sequence.

More precisely, let ρ and η be respectively the number of positive and negative values, the maximum number of time a change in sign can occur, denoted by τ, is

$$\tau = \begin{cases} 2p-1 & \text{if } p=n \\ 2\min(p,n) & \text{otherwise} \end{cases} \tag{3.4}$$

and let δ be the number of sign changes observed, then

$$\text{ALPHA}_{\text{transaction}} = \frac{\delta}{\tau} \tag{3.5}$$

is an indicator of how often “ping pong” exchanges are seen in a flow. τ is equal to 0 when the flow is unidirectional and thus $\text{ALPHA}_{\text{transaction}}$ may not be defined for all flows. $\text{ALPHA}_{\text{transaction}}$ is initialized to zero by default. When $\text{ALPHA}_{\text{transaction}}$ is non-zero, a value close to 1 is a strong indicator of multiple transaction exchanges.

3.4.3. Encryption Indicators

Encryption indicators are calculated by comparing the greatest common divisor (GCD) values of payloads.

The algorithm follows an iterative process. At each step, the array of input is broken into two parts for pair wise GCD calculation, and the array to be examined in the following step will contain the GCD values that are greater than 1.

The process is interrupted if, at a given step, the count of GCDs that are greater than 1 is smaller than the count of GCDs equalled to 1. The calculation is done for each direction separately, the output gives two values:

$\text{ALPHA}_{\text{cipherblock}}$ gives the estimated popular GCD among payload lengths of packets. $\text{BETA}_{\text{cipherblock}}$ gives the ratio of non-empty packet-payloads that are divisible by $\text{ALPHA}_{\text{cipherblock}}$.

If the GCD calculation process got interrupted due to too many pair-wise GCD equal to one, then the value for $\text{ALPHA}_{\text{cipherblock}}$ is equal to 1 and the value for $\text{BETA}_{\text{cipherblock}}$ is set to 0.

The reason of evaluating the GCD values for encryption related packets is that the lengths of encrypted packets typically have a greatest common divisor different than one (Zhang and Paxson 2000).

3.4.4. Command Line and Keystroke Indicators

Command-line transmissions are larger in size and are separated by longer delays than keystrokes. The distinction between command-line and keystroke interactivity helps refine the classification process a step further. FTP command for instance can be distinguished from interactive SSH and TELNET sessions; and it is foreseen that chat sessions will be classed differently depending on the "flavour" (De Montigny and Leboeuf 2005).

For keystroke interactive indicators, a small packet is defined as a non-empty packet when carrying 60 bytes or less. The inter-arrival delays between keystrokes are taken as between 25ms (d_{min}) and 3000ms (d_{max}).

On the other hand, for command line indicators, a small packet is defined as a non-empty packet when carrying 200 bytes or less. The inter-arrival delays between keystrokes are taken as between 250ms and 30000ms.

For each direction of the flow, let Ω be the set of delays between consecutive small packets and $\Delta = \{ \omega \in \Omega$, such that $dmin \leq \omega \leq dmax \}$, the indicator of interactive inter-packet departure is defined as:

$$\text{ALPHA}_{\text{interactive}} = \frac{\textit{number of elements} \in \Delta}{\textit{number of elements} \in \Omega} \tag{3.6}$$

Let S be the number of small packets, let N be the number of non-empty packets, let G be the number of gaps between small packets, the indicator of interactivity based on the proportion of small packets is:

$$\text{BETA}_{\text{interactive}} = \frac{S}{N} \tag{3.7}$$

Here, a gap occurs whenever two small non-empty packets are separated by at least one packet (big or empty). The indicator of consecutive small packets is

$$\text{GAMMA}_{\text{interactive}} = \frac{S-G-1}{N} \tag{3.8}$$

The fourth indicator gives the proportion of small non-empty packets with respect to the total number of small packets (including empty packets). The goal with this heuristic is to penalize machine-driven applications that transmit a lot of small packets, which may however be dominated by empty control segments (i.e. TCP ACK packets without piggyback data). Thus let E be the number of empty packets, it is defined:

$$\text{DELTA}_{\text{interactive}} = \frac{S}{S+E} \tag{3.9}$$

Lastly, it is defined a fifth indicator measuring irregularity in the transmission rate of consecutive small packets. Let μ and σ be respectively the mean and standard deviation of the delays between consecutive small packets; let Λ= { ω ∈ Ω, such that ω ∈ [μ-σ,μ+σ] }, then the indicator of irregularity between inter-arrival times of consecutive small packets is:

$$\text{EPSILON}_{\text{interactive}} = 1 - \frac{\textit{number of elements} \in \Lambda}{\textit{number of elements} \in \Omega} \tag{3.10}$$

3.4.5. File Transfer Indicators

From the interactive indicators, it is derived file transfer indicators. In general, a file transfer flow contains episodes of consecutive big packets transmitted within a short delay. A big packet is defined as carrying 225 or more bytes. A short inter-packet delay is 50ms or less.

For each direction of the flow, let B be the number of big packets, let N be the number of non-empty packets, let G' be the number of gaps between big packets. Furthermore, let Ω' be the set of delays between consecutive big packets and Δ' = { ω ∈ Ω', such that ω ∈ [0, dmax] }, then the indicator of inter-packet departure during a file transfer is:

$$\mathrm{ALPHA}_{\mathrm{file}} = \frac{number\ of\ elements \in \Delta'}{number\ of\ elements \in \Omega'} \tag{3.11}$$

The indicator of file transfer based on the proportion of big packets is defined as:

$$\mathrm{BETA}_{\mathrm{file}} = \frac{B}{N} \tag{3.12}$$

and lastly, the indicator of consecutive big packets is

$$\mathrm{GAMMA}_{\mathrm{file}} = \frac{B - G' - 1}{N} \tag{3.13}$$

3.5. Deriving Anomaly

According to the *Lakhina*, *Crovella* and *Diot's* work (Lakhina, Crovella and, Crovella 2004), number of bytes, number of packets and number of IP flow values for a flow traffic can be used to identify anomalies on the network traffic. It is required the evaluation of multivariate time series of origin-destination flow traffic defined as # of bytes, # of packets and # of IP flows. By using the subspace method (Lakhina, Crovella, and Dio 2004) it is showed that each of these traffic types reveals a different (sometimes overlapping) class of anomalies and so all three types together are important for anomaly detection.

The subspace method works by examining the time series of traffic in all OD flows simultaneously. It then separates this multivariate timer series into normal and anomalous attributes. Normal traffic behavior is determined directly from the data, as the temporal patterns that are most common to the ensemble of OD flows. This extraction of common trends is achieved by Principal Component Analysis (Lakhina, Crovella and, Crovella 2004).

To produce the multivariate time series of a flow traffic, splitting the record is required. Rasplit, Argus client, is used for splitting the record into sub flows depending on a time interval. The time interval is chosen depending on the flow duration. Either 1 minute or 5 minute time interval values are used.

By traversing on the sub flow files, # of bytes, # of packets or # of IP flow values are calculated depending on the origin-destination pairs. A file with comma separated values (CSV) is created. The CSV file is used for creating matrix for PCA evaluation. After the PCA value is calculated, transformed data of the PCA result is graphed. It is observed that peeks at the graphs represents the anomalies at the network traffic.

CHAPTER 4

IMPLEMENTATION

This section includes the implementation details of the approach preferred in this thesis. Starting from the programming language preferred from development, class structure details and the implementation steps are explained. Python is chosen as the development programming language. A class hierarchy is constructed and implemented by using Python. The details of the class hierarchy is given as UML diagrams. By following the class structure and using the Python language, steps that should be followed for running the software is defined.

4.1. Programming Language

This thesis can be divided into two level of survey. The first level is the traffic characterization, which requires pcap file traversing to get the packet level information, operation on MySQL database tables and mathematical calculations. Second level is the PCA part which is done mainly by Octave. It is used a CSV file for Octave's *princomp* function as an input. The CSV file creation requires flow splitting and traversing on the readable flow files to calculate # of values and save them to database.

It is chosen Python as the programming language for development. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for rapid application development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed (Python 2009). Python is also actively used by penetration testers, vulnerability researchers and information security practitioners.

Python's Scapy (Scapy 2009) module is used to decode packets from pcap files and analyze them according to their header information. Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy is mainly used for traversing through the prerecorded network traffic via tcpdump and get IP level information from packets for saving them to the database.

By using Scapy it is possible to get the source IP address, source port number, destination IP address, destination port number, packet size, payload ingredients and payload size, capture time and protocol information from each packet. Capture time is valuable for calculating jitter. Payload size is important for detecting the non-empty packets. The rest of the information that is 5-tuple key also, is used to understand what flows they belong to.

4.2. Class Structure

4.2.1. Flow Object

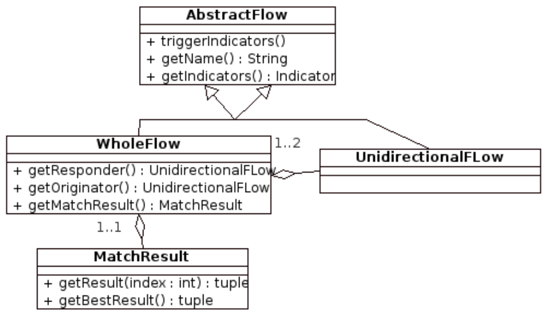

AbstractFlow
+ triggerIndicators()
+ getName() : String
+ getIndicators() : Indicator
WholeFlow
+ getResponder() : UnidirectionalFLow
+ getOriginator() : UnidirectionalFLow
+ getMatchResult() : MatchResult
1..2
UnidirectionalFLow
1..1
MatchResult
+ getResult(index : int) : tuple
+ getBestResult() : tuple

Figure 4.1. AbstractFlow Diagram

Every flow is defined as an object. As it can be seen from Figure 4.1 (Gerçek 2009), it is defined two types of flows. One type is the directional flow. It is defined an UnidirectionalFlow class to create directional flows. A directional flow may be either from source IP number (Originator) to destination IP number (Responder) or vice versa. WholeFlow class is inherited from an AbstractFlow class. AbstractFlow is a general class that has abstract definitions to create flow indicator objects.

Flows are created by using a generator class.

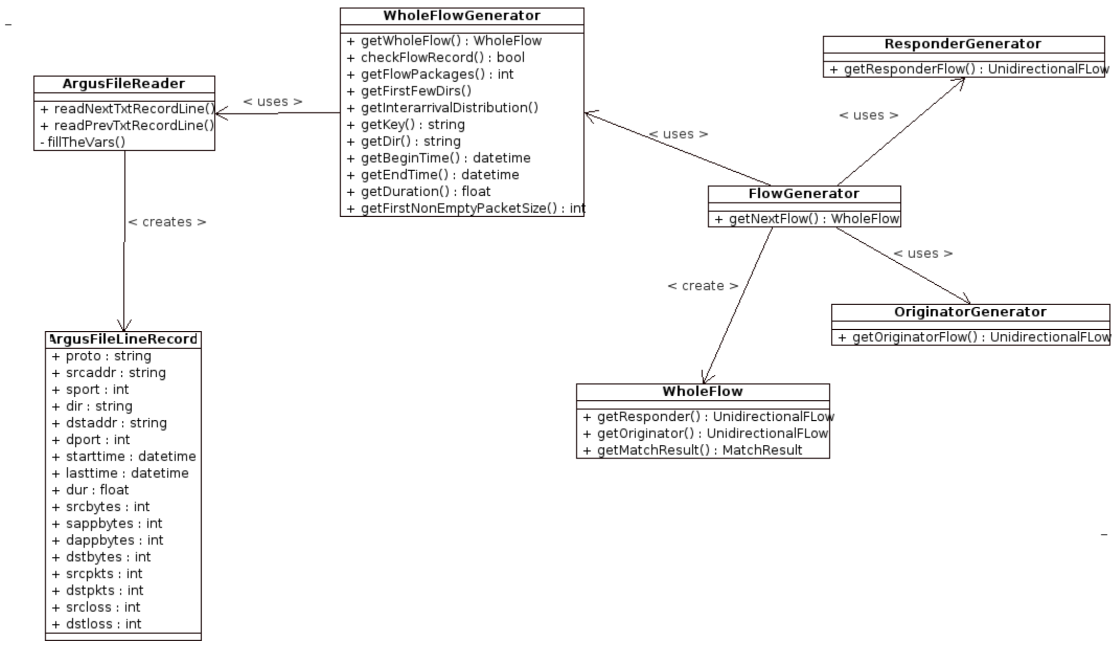


Figure 4.2. FlowGenerator and WholeFlowGenerator Class Relations

It is required to calculate attributes related with a whole flow. Attribute related details are given at Figure 4.2 (Gerçek 2009). By using ArgusFileReader class, every line of the produced flow text record is traversed and whole flow attributes are calculated.

Except from the attributes mentioned inside the WholeFlowGenerator, additional directional and whole flow indicators are required to be calculated. To achieve indicator calculation, generators are used. Figure 4.3 presents the relation between indicators.

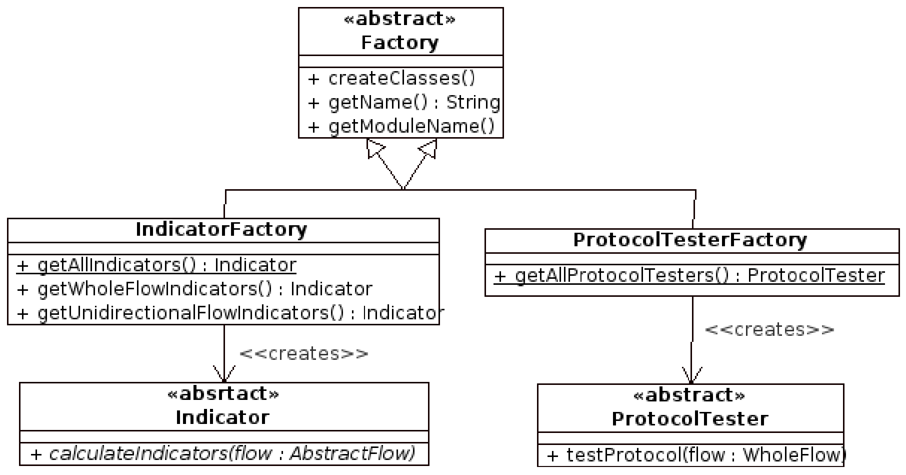

«abstract»
Factory
+ createClasses()
+ getName() : String
+ getModuleName()
IndicatorFactory
+ getAllIndicators() : Indicator
+ getWholeFlowIndicators() : Indicator
+ getUnidirectionalFlowIndicators() : Indicator
ProtocolTesterFactory
+ getAllProtocolTesters() : ProtocolTester
<<creates>>
<<creates>>
«absrtact»
Indicator
+ calculateIndicators(flow : AbstractFlow)
«abstract»
ProtocolTester
+ testProtocol(flow : WholeFlow)

Figure 4.3. Factory Class Diagram

At Figure 4.3 (Gerçek 2009), IndicatorFactory includes the method calls for each flow object handled.

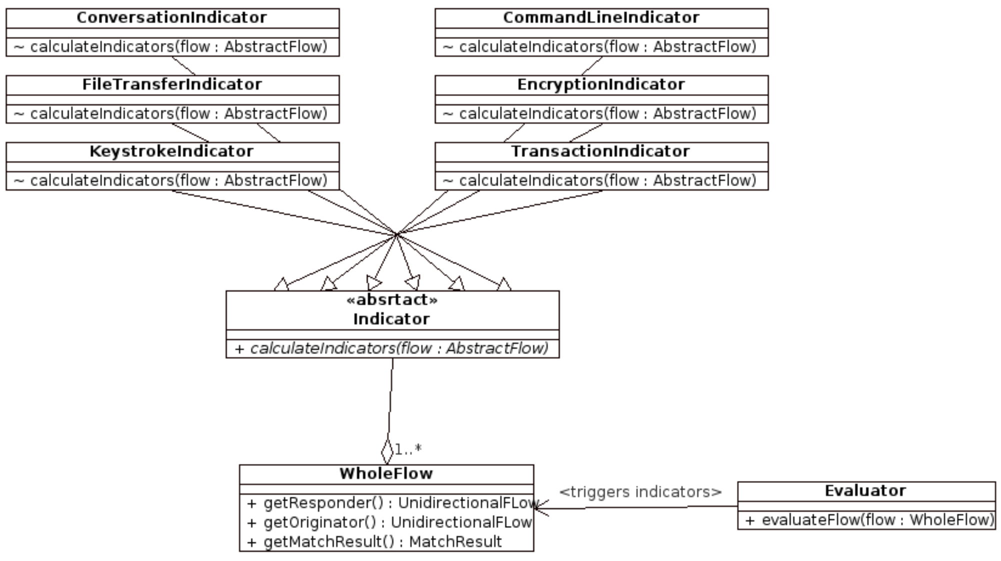
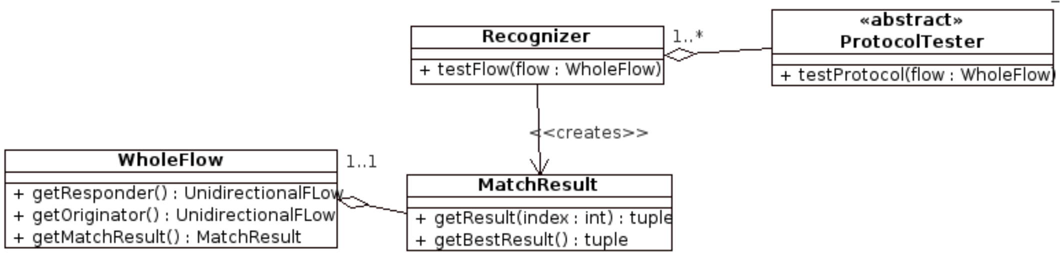

ConversationIndicator
~ calculateIndicators(flow : AbstractFlow)
CommandLineIndicator
~ calculateIndicators(flow : AbstractFlow)
FileTransferIndicator
~ calculateIndicators(flow : AbstractFlow)
EncryptionIndicator
~ calculateIndicators(flow : AbstractFlow)
KeystrokeIndicator
~ calculateIndicators(flow : AbstractFlow)
TransactionIndicator
~ calculateIndicators(flow : AbstractFlow)
«absrtact»
Indicator
+ calculateIndicators(flow : AbstractFlow)
1..*
WholeFlow
+ getResponder() : UnidirectionalFLow
+ getOriginator() : UnidirectionalFLow
+ getMatchResult() : MatchResult
<triggers indicators>
Evaluator
+ evaluateFlow(flow : WholeFlow)

Figure 4.4. Indicator Class Diagram (Gerçek 2009)

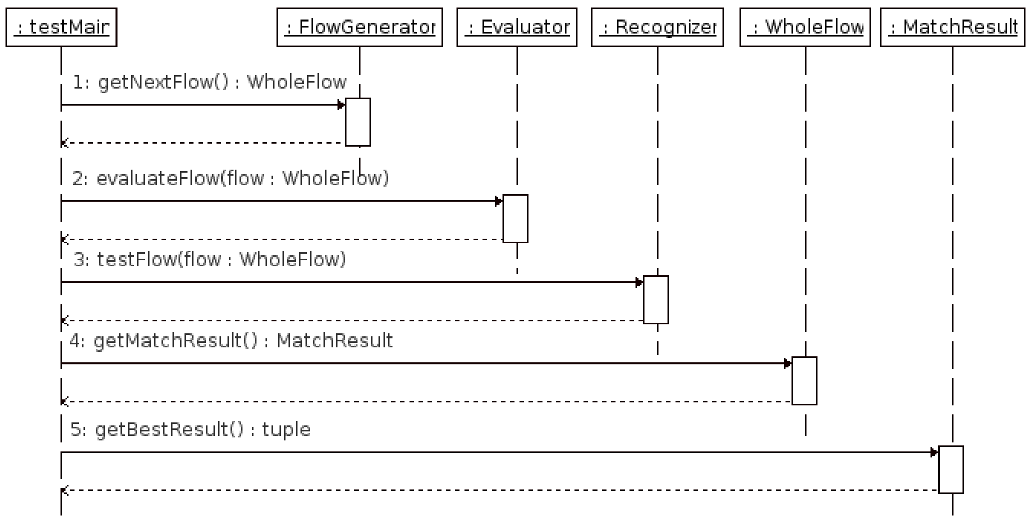
The mathematical calculations are done inside the each of the indicators mentioned. Each of them are presented at Figure 4.4 (Gerçek 2009), above. The aim of calculating attribute values is to decide about the protocol. By looking at the results, the flow examined is tried to be categorized as an application level protocol. ProtocolTester includes the match cases that should be taken into consideration for each flow.



Recognizer
+ testFlow(flow : WholeFlow)
1..*
«abstract»
ProtocolTester
+ testProtocol(flow : WholeFlow)
<<creates>>
WholeFlow
+ getResponder() : UnidirectionalFLow
+ getOriginator() : UnidirectionalFLow
+ getMatchResult() : MatchResult
1..1
MatchResult
+ getResult(index : int) : tuple
+ getBestResult() : tuple

Figure 4.5. Recognizer Class Diagram

As a result of attribute calculation, a match result is calculated for each flow examined. The result includes the best 3 match. The relation between match result calculation and recognizer is mentioned at Figure 4.5 (Gerçek 2009), as above.



: testMain
: FlowGenerator
: Evaluator
: Recognizer
: WholeFlow
: MatchResult
1: getNextFlow() : WholeFlow
2: evaluateFlow(flow : WholeFlow)
3: testFlow(flow : WholeFlow)
4: getMatchResult() : MatchResult
5: getBestResult() : tuple

Figure 4.6. Threshold Calculation Sequence Diagram

For a general view, lets check the Figure 4.6. For each of the flow generated a testFlow class is used to trigger the match results. By looking at the best match results a threshold for each of the protocol is decided.

Any flow that does not have a best result above the threshold is save as an undefined flow for further evaluation. The PCA process is explained as follows:

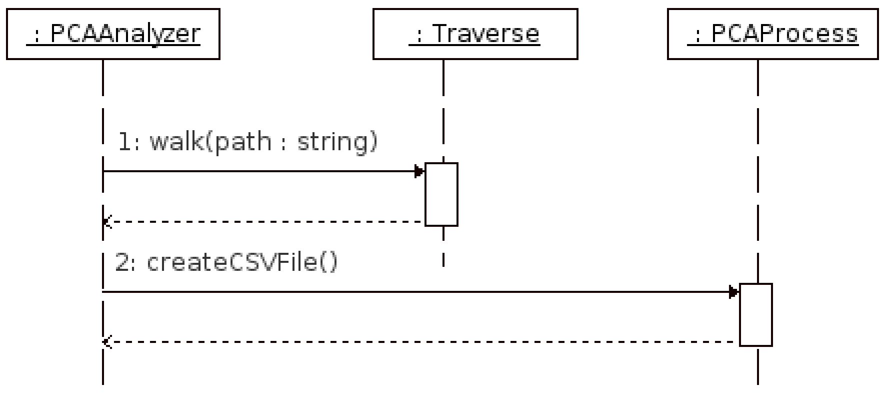


Figure 4.7. PCA Sequence Diagram

Packages related with the undefined flows are saved and splitted for PCA analysis. The splitted pcap files are traversed and saved to a database table. After this process the table is used to produce a CSV file for PCA graph operation.

4.3. Software Structure

The developed software takes a pcap record as input. After the following steps, the result is a graph which may include peeks. The whole is not fully automatized.

1. pcap file is traversed and saved to the database: This process is the requirement part of all the processes. Depending on the captured packet size time varies.
2. Human readable flow information is saved into the text file: This process

is done via using Argus and racluster commands.

3. UndefinedTest.py is run: The name of the produced flow text file is changed before running, so the file is used for per flow attribute calculation. Each line of the text file includes one flow record. For each flow record, attribute values are calculated and a match result is returned. A typical match result include the best three matches:

```
l.      [(0.65, 'FTPCommand'),(0.80, 'Telnet' ),(1.0, 'SSH')]
```

By looking at the threshold values and the highest match percentage, the flow is categorized. If it is not possible to categorize the flow it is save as undefined for PCA process. What is saved is mainly the key values of the flow. Below table gives the details:

Table 4.1. Table structure for table UndefinedFlowTbl

Field	Type	Null	Default
orderno	bigint(20)	Yes	NULL
srcip	varchar(20)	Yes	NULL
dstip	varchar(20)	Yes	NULL
dstport	int(11)	Yes	NULL
sport	int(11)	Yes	NULL
proto	varchar(10)	Yes	NULL
begintime	varchar(60)	Yes	NULL

4. A new pcap file is created: After the undefined flow detection is finished, packages that belong to the undefined flows should be separated and saved as a new undefined pcap file. For this process a Perl script called pcap-util (Boddington 2009) is used. Pcap-util enables libpcap (Libpcap 2009) filter language usage, that is prerecorded data can be filtered according host, port, time information or some other parameters. It is used the IP address filtering as follows:

```
m.      ./pcap-util filter infile.dump outfile.dump "host 192.168.1.118 or  host
        192.168.1.188"
```

host parameter is used to define source and destination IP addresses, so above command finds and extracts the packets related with the above IP addresses and saves them as a outfile.dump pcap file.

5. <u>outfile.dump file is splitted into time series</u>: The pcap file is first converted to a flow data and then rasplit is called to split the file into sub flows:

n. rasplit -r outfile.dump.pcap.arg3 -M time 1m -w argus-%d.%m.%Y-%H:%M:%S – ip

Above command splits the flow file into 1 minute sub flows. Each sub flow is converted to a readable text file for being processed.

6. <u>traverseAndSave.py module is used</u>: This module is walk through the directory that contains sub flow text files and saves the origin-destination related # of bytes and # of packet values to the database table. PCAProcess.py is then used to create a CSV file for PCA process.
7. <u>Octave is run</u>: By using the CSV file, a matrix is created first.

o. x=dlmread ("graph.csv",",");

The created X matrix is used for PCA calculation:

p. [a,b,c,d]=princomp(x);

b value calculated at the (36) above includes the transformed data. It is used for plotting. By using the plot function, anomaly graph is produced.

CHAPTER 5

TEST RESULTS

This chapter is dedicated for the test results that are processed to calculate threshold values for each application level protocol. Calculating threshold results are an important step for anomaly detection. By looking at the threshold values, undefined traffic is decided. This chapter also includes PCA results. PCA is first tested on prerecorded data, then applied on manually produced attacks.

5.1. PCA Tests

5.1.1. Tests with Clean Traffic

PCA process is first tested on the clean traffic. 1999 DARPA Intrusion Detection Evaluation Data Set's week1 day1 and week 5 day 2 recorded traffic is used. It is known that week 1 has a clean traffic though week 5 has labeled attacks. So the produced graphs can be used for comparison and understanding the peek concept.

Week 1 day 1 record is splitted into 5 minutes sub flows and throughout these sub flows, origin-destination pair of # of packages values are calculated and graphed by using Octave's *princomp* function. The resulted graph is as follows:

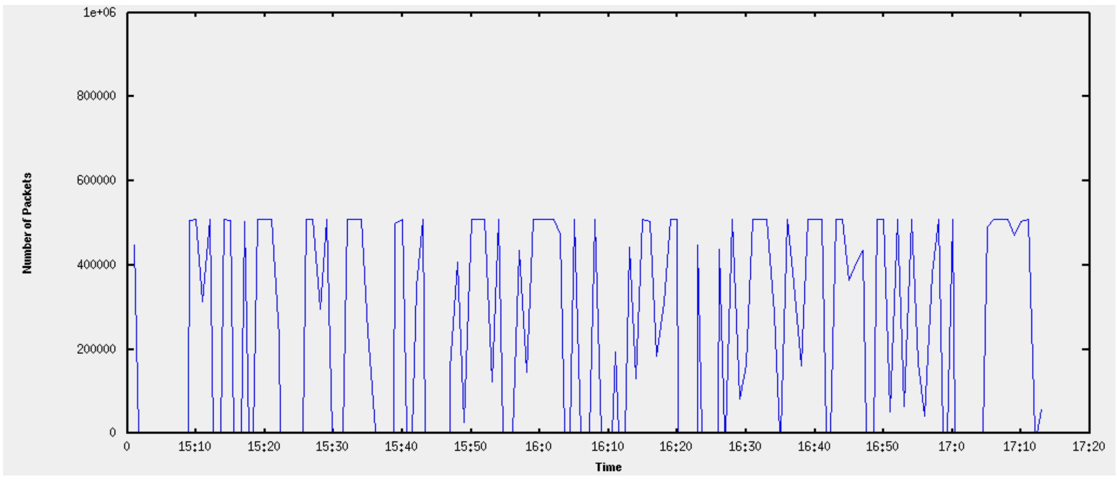


Figure 5.1. PCA Analysis of Week 1 Day 1 Record

As it can be seen from Figure 5.1, there is no sudden increase at the graph levels. The same process is done for the week 5 day 2 record and the below graph is produced.

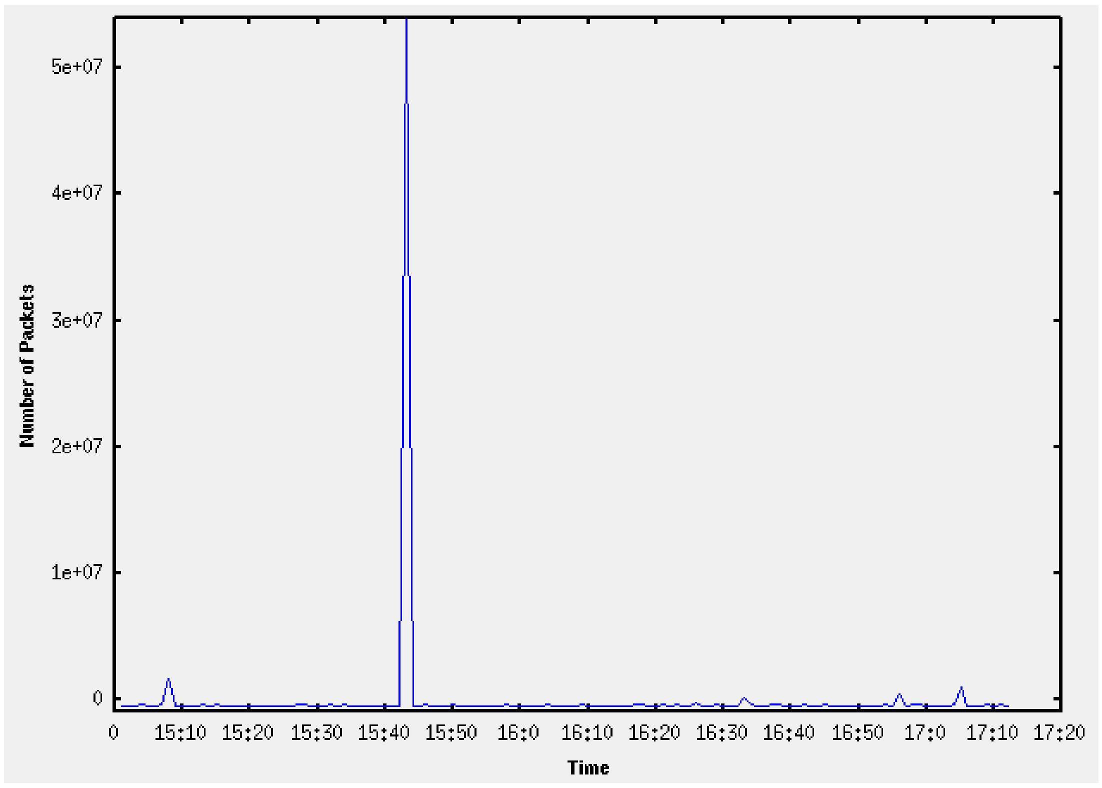


Figure 5.2. PCA Analysis of Week 5 Day 2 Record

When Figure 5.1 and Figure 5.2 is compared, it can be seen that the later one has sudden increases at the y levels. It is mentioned that these peeks correspond to anomalies at the network traffic.

5.1.2. Tests with Manual Attacks

By using ettercap DOS attack plugin, manually produced DOS attack is produced. At the target machine, a regular Internet traffic is produced, like web surfing, MSN Chat and file downloading. During these traffics and attack, the traffic is saved and analyzed. Below is the information related with collected traffic:

q.
```
File name: ettercap-dos.pcap
File type: Wireshark/tcpdump/... - libpcap
File encapsulation: Ethernet
Number of packets: 5495141
File size: 415279995 bytes
Data size: 332851811 bytes
Capture duration: 307.992800 seconds
Start time: Mon May  4 15:19:30 2009
End time: Mon May  4 15:24:38 2009
Data rate: 1080712.96 bytes/s
Data rate: 8645703.69 bits/s
Average packet size: 60.57 bytes
```

After the traffic characterization, undefined flows are detected and by using the pcap-util, a new pcap file is created. When the PCA is processed over the new pcap file the following graph is produced.

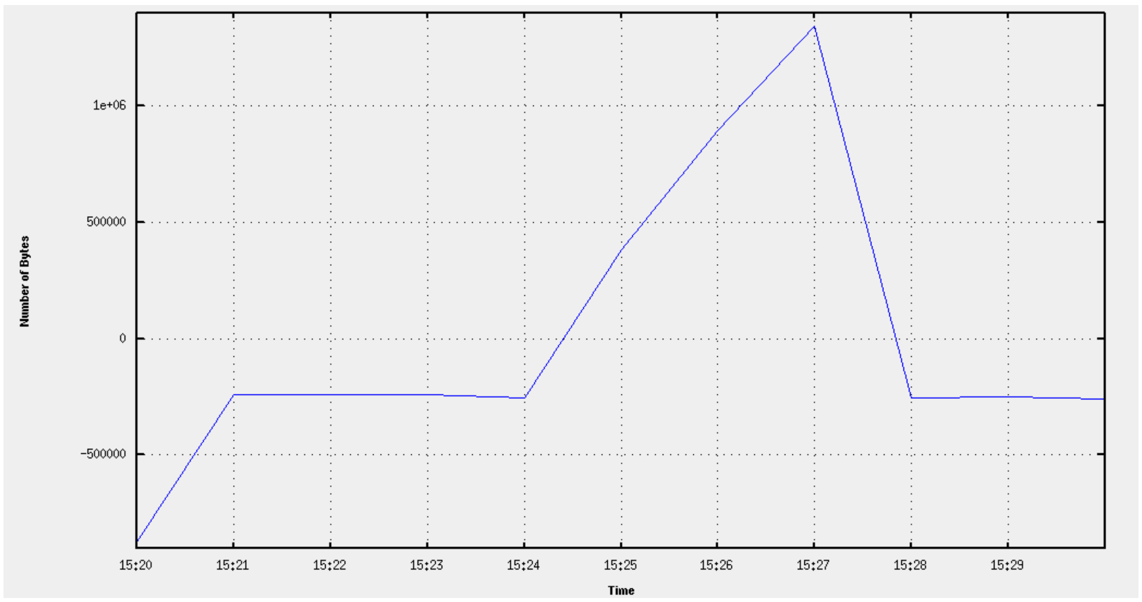
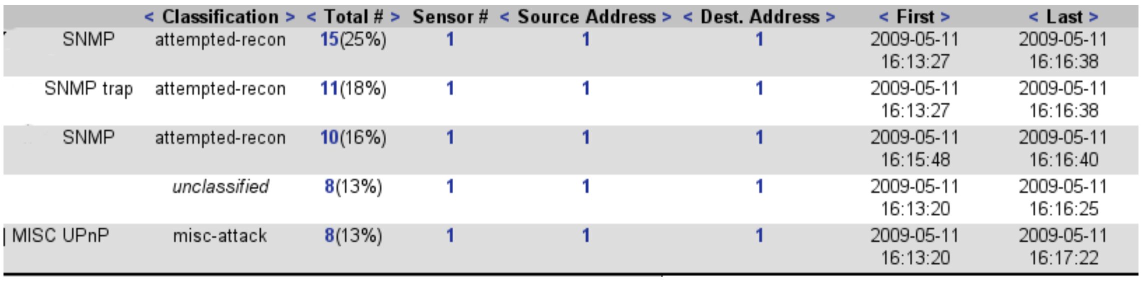


Figure 5.3. Ettercap DOS Attack PCA Analysis

It can be observed from Figure 5.3 that between the time intervals 15:24 and 15:28, there is an anomaly at the network traffic. When the flows are checked and the packets belong to these flows are saved as another pcap file, the new pcap file is used for tcpreplay program. By using the tcpreplay and Snort, the suspicious traffic is rechecked. Tcpreplay enables reproducing the pcap data through an Ethernet interface.

Below is the Snort result that is taken from the Snort BASE interface.



	< Classification >	< Total # >	Sensor #	< Source Address >	< Dest. Address >	< First >	< Last >
SNMP	attempted-recon	15(25%)	1	1	1	2009-05-11 16:13:27	2009-05-11 16:16:38
SNMP trap	attempted-recon	11(18%)	1	1	1	2009-05-11 16:13:27	2009-05-11 16:16:38
SNMP	attempted-recon	10(16%)	1	1	1	2009-05-11 16:15:48	2009-05-11 16:16:40
	unclassified	8(13%)	1	1	1	2009-05-11 16:13:20	2009-05-11 16:16:25
| MISC UPnP	misc-attack	8(13%)	1	1	1	2009-05-11 16:13:20	2009-05-11 16:17:22

Figure 5.4. Snort Result of Undefined Traffic, DOS Attack

The same test is done for a scan attack. Nmap is used for scanning the target as follows:

r. nmap -P0 -sS -p0-65535 -e eth0 192.168.1.188

When the sniffed traffic is analyzed as above, the resulted graph is below:

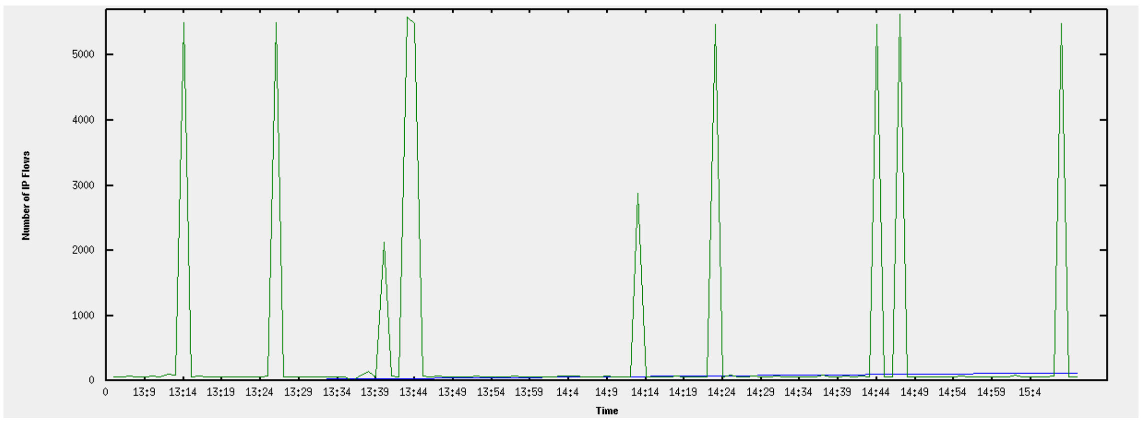


Figure 5.5. Nmap Syn Scan PCA Analysis

It can be seen from Figure 5.5 that, it is used # of IP flows for the origin-destination pairs to gather scan results.

5.2. Threshold Tests

It is mentioned that threshold values are necessary to decide undefined flows. Below are the tested values and the duration of the calculation. The tests are run over 100 flow records. Every test is done for a dedicated traffic, that is FTPCommand is tested with the traffic that includes only flows with port number 21.

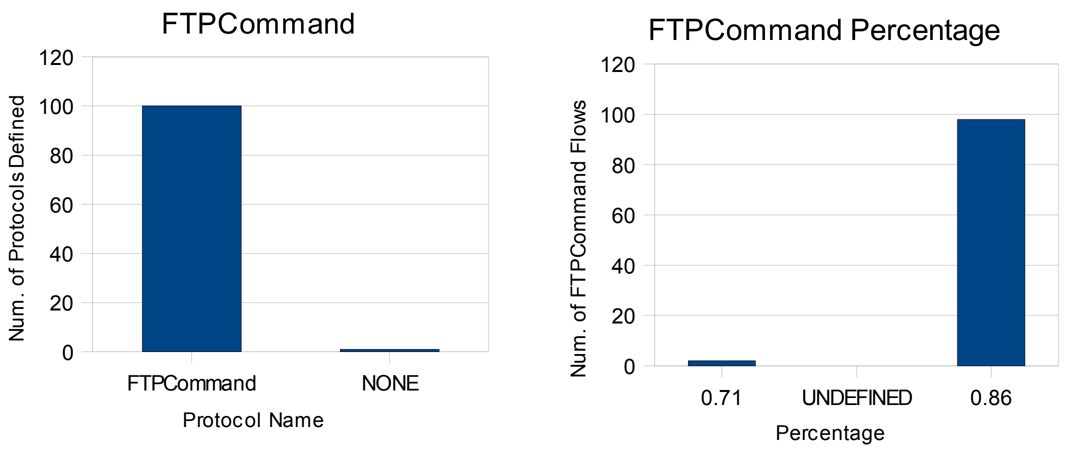


Figure 5.6. FTPCommand Test Results

For 100 FTPCommand related flows, every of them is defined as FTPCommand. 2 of the defined FTPCommand flows are matched with 71%, and the rest 98 flows are matched with 98%. Total duration time of the calculation is 79 minutes.

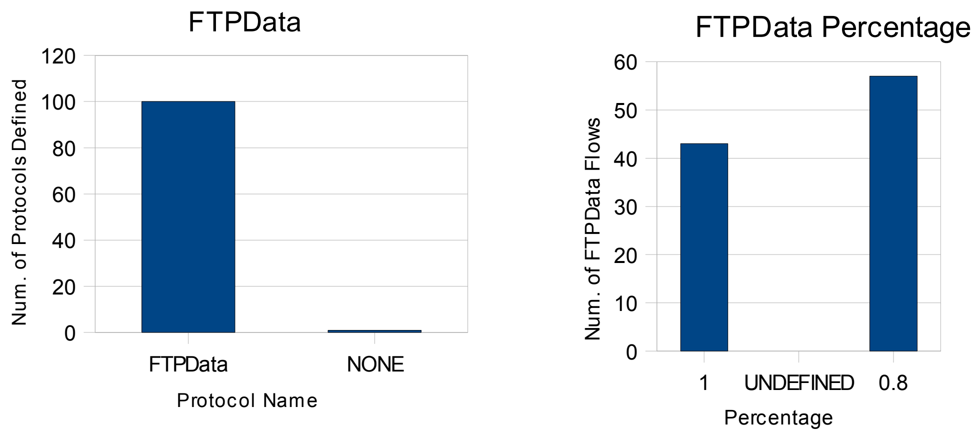


Figure 5.7. FTPData Test Results

For 100 FTPData related flows, every of them is defined as FTPCommand. 43 of the defined FTPData flows are matched with 100%, and the rest 57 flows are matched with 80%. Total duration time of the calculation is 72 minutes.

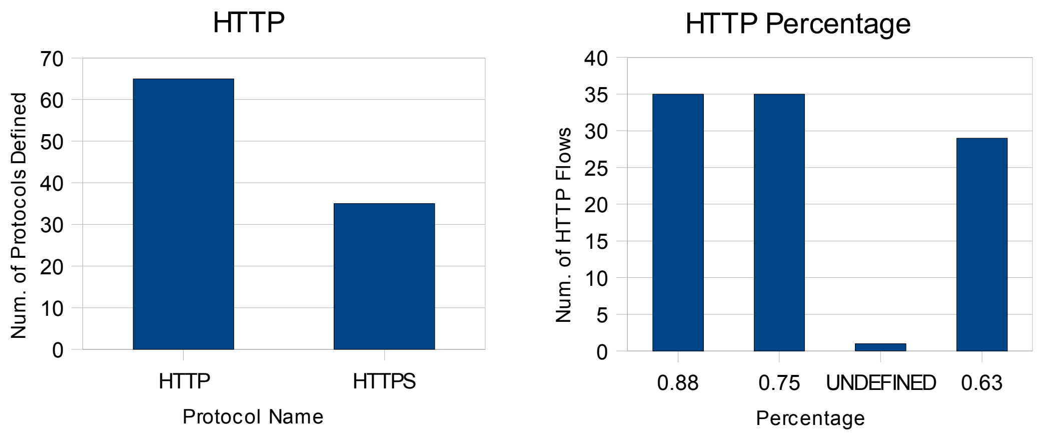


Figure 5.8. HTTP Test Results

For 100 HTTP related flows, 65 of them is defined as HTTP. 35 of them is defined as HTTPS, 35 of the defined HTTP flows are matched with 88%, another 35 of them are matched with 75%, one of them is not matched and the rest 29 flows are matched with 63%. Total duration time of the calculation is 47 minutes.

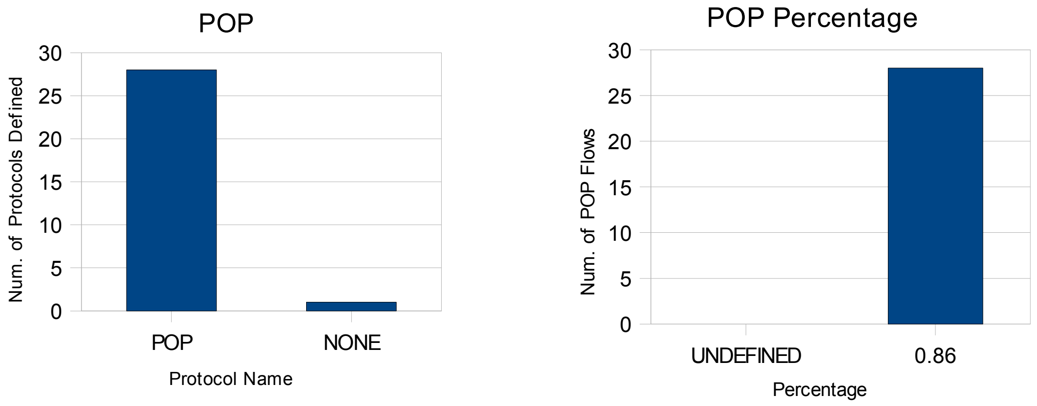


Figure 5.9. POP Test Results

For 28 POP related flows, except from one of them every of them is defined as POP. All the defined POP traffic is matched with 86%. Total duration time of the calculation is 22 minutes.

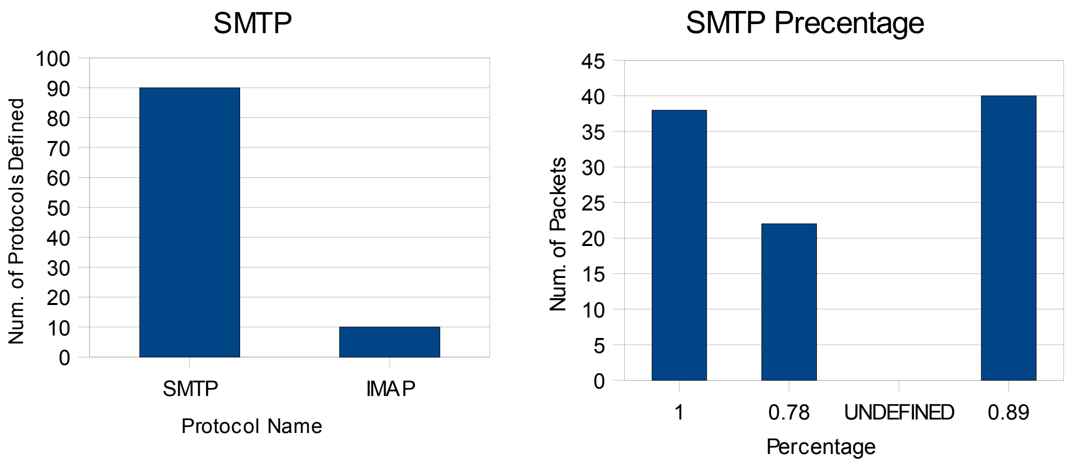
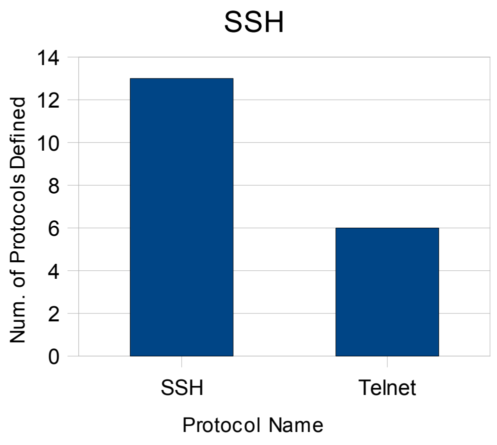


Figure 5.10. SMTP Test Results



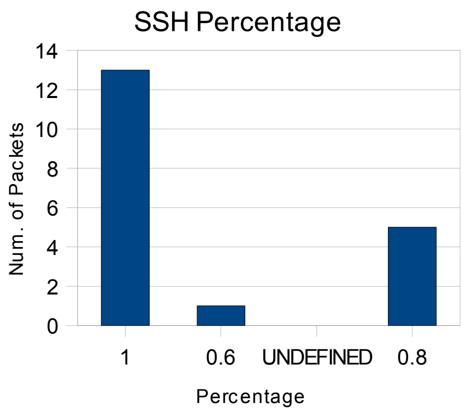
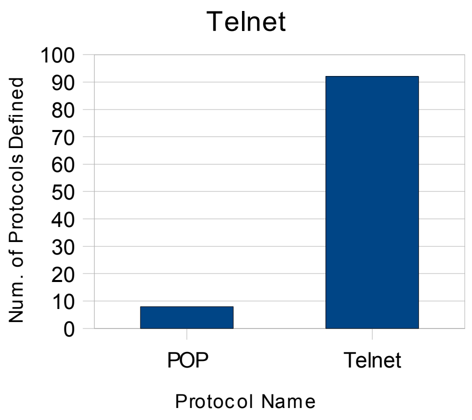


Figure 5.11. SSH Test Results

For 100 SMTP related flows, 90 of them is defined as SMTP. 10 of them is defined as IMAP, 38 of the defined POP flows are matched with 100%, another 22 of them are matched with 78%, and the rest 40 flows are matched with 89%. Total duration time of the calculation is 90 minutes.

For 19 SSH related flows, 13 of them is defined as SSH. 6 of them is defined as Telnet, 13 of the defined SSH flows are matched with 100%, another 1 of them are matched with 60%, and the rest 5 flows are matched with 80%. Total duration time of the calculation is 25 minutes.



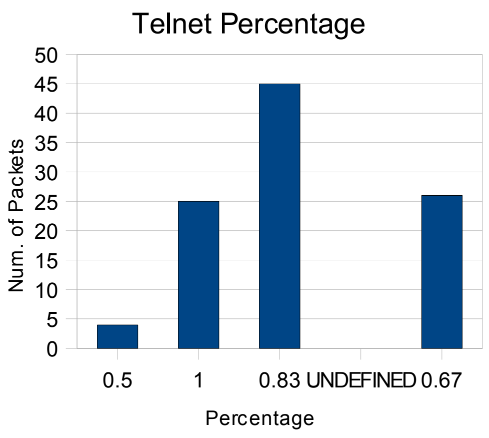


Figure 5.12. Telnet Test Results

For 100 Telnet related flows, 8 of them is defined as POP. 92 of them is defined as Telnet, 4 of the defined Telnet flows are matched with 50%, another 25 of them are matched with 25%, 45 of them are defined with 83% and the rest 26 flows are matched with 67%. Total duration time of the calculation is 92 minutes.

5.3. BitTorrent Traffic Analysis

Flow attributes for each flow are calculated depending the predefined criteria. The criteria defined at the *De Montigny* and *Leboeuf (De Montigny and Leboeuf 2005)* don't include BitTorrent traffic restrictions. In this thesis it is aimed to calculate the required criteria to characterize BitTorrent traffic.

5.3.1. BitTorrent Protocol

The BitTorrent Protocol (BTP) is a protocol for collaborative file distribution across the Internet and has been in place on the Internet since 2002. It is best classified as a peer-to-peer (P2P) protocol, although it also contains highly centralized elements. BTP has already been implemented many times for different platforms, and could well be said to be a mature protocol (Fonseca and Reza 2005).

The protocol works when a file provider initially makes his/her file (or group of files) available to the network. This is called a *seed* and allows others, named *peers*, to connect and download the file. Each peer that downloads a part of the data makes it available to other peers to download. After the file is successfully downloaded by a peer, many continue to make the data available, becoming additional seeds. This distributed nature of BitTorrent leads to a viral spreading of a file throughout peers. As more peers join the swarm, the likelihood of a successful download increases. Relative to standard Internet hosting, this provides a significant reduction in the original distributor's hardware and bandwidth resource costs. It also provides redundancy against system problems and reduces dependence on the original distributor (BitTorrent 2009).

5.3.2. Testbed for BitTorrent Characterization

To characterize torrent traffic, a clean traffic is required to be analyzed. To get a clean BitTorrent traffic, torrent traffic is manually collected. To achieve this goal, all the network services at a host on a network is closed and by using a torrent client tool (Ktorrent (Ktorrent 2009) is used on Kubuntu Linux system), all the traffic is saved in 100MB parts by using Tcpdump on the same network interface.

The first 100MB part is used to decide the traffic protocol limits. First the pcap file is traversed and saved on MySQL database. After converted to Argus flow record , protocol tester modules are run and for a torrent traffic, the below similarities are gathered:

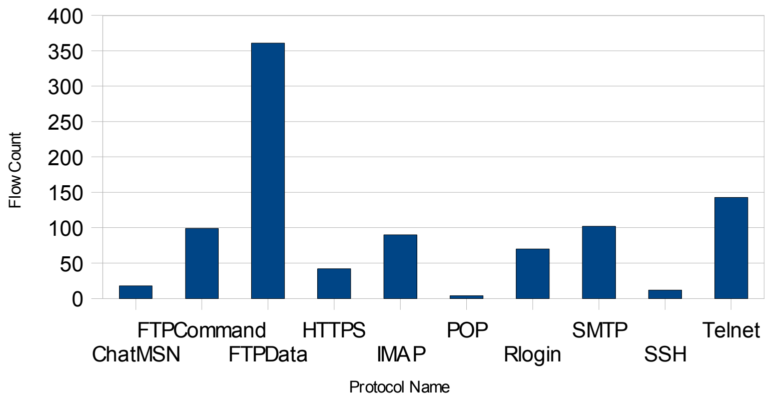


Figure 5.13. Protocol Similarities Defined for Torrent Traffic

From the Figure 5.6 it can be seen that BitTorrent traffic is more like a FTPData, Telnet and FTPCommand. Indeed BTP consists of two logically distinct protocols, namely the Tracker HTTP Protocol (THP), and the Peer Wire Protocol (PWP). THP defines a method for contacting a tracker for the purposes of joining a swarm, reporting progress etc. PWP defines a mechanism for communication between peers, and is thus responsible for carrying out the actual download and upload of the torrent (Fonseca and Reza 2005).

By observing the HTTP, FTPData, FTPCommand and Telnet protocol's recognizer criteria, characteristics are tried to be defined for BitTorrent traffic. 1000 BitTorrent flows are used and their attribute values are saved to database.

5.3.3. Criteria for BitTorrent

The first handled attribute is payload distribution. By using the packet based queries on the MySQL table, it is seen that Originator payload distribution has a meaningful characteristics.

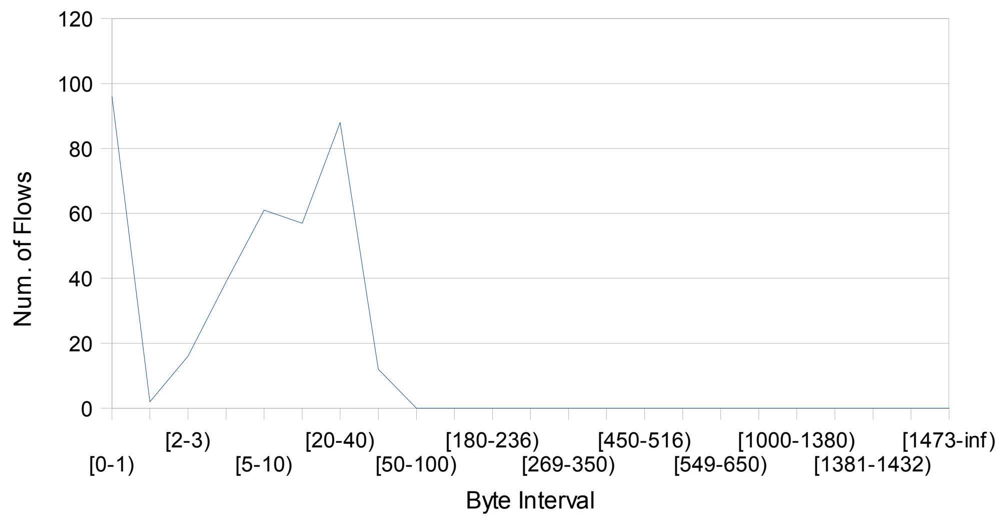


Figure 5.14. Payload Distribution Graph for Originator

According to Figure 5.7, there is no flow that has a payload distribution over 50 bytes. So for torrent traffic, source flows will have payload distribution values less than 50 bytes.

Second handled criteria is DatabyteRatioOrigToResp value of the whole flow attribute.

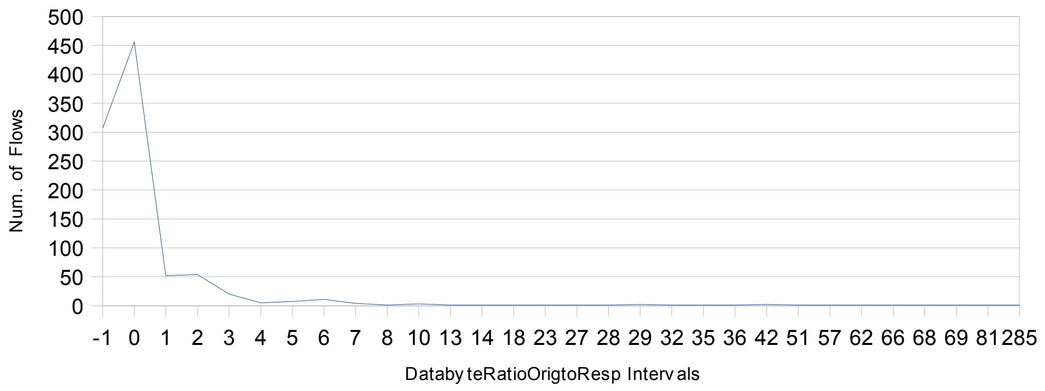


Figure 5.15. DatabyteRatioOrigToResp Distribution Graph

It can be seen from the Figure 5.8 that ratio of the total number of data bytes sent by originator over total number of data bytes sent by responder is always less than 4. Because torrent traffic is a data transaction also, handling DataByteCount and ByteCount values will be meaningfully. Figure 5.9 and Figure 5.10 shows the DataByteCount value over ByteCaount value for each directional flow.

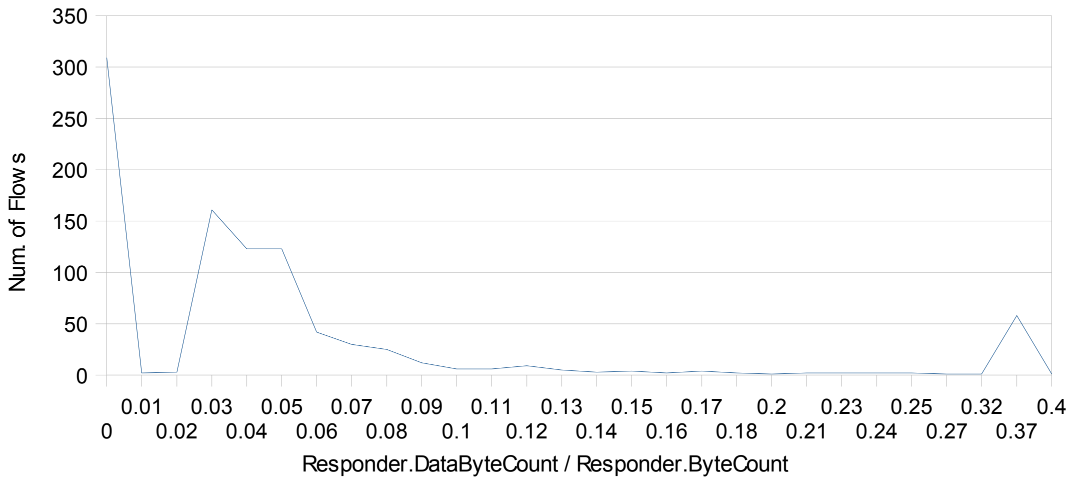


Figure 5.16. DataByteCount/ByteCount Graph for Responder

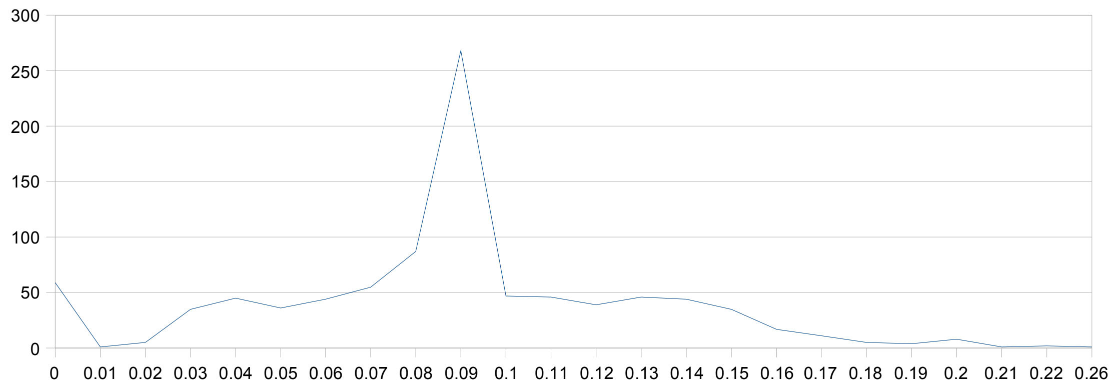


Figure 5.17. DataByteCount/ByteCount Graph for Originator

From Figure 5.9 it can be seen that (Responder.DataByteCount / Responder.ByteCount) value is less than 0.40 though (Originator.DataByteCount / Originator.ByteCount) value at the Figure 5.10 is less than 0.27.

Another attribute value that is taken into consideration is firstNonEmptyPacketSize. According to the Figure 5.11 the value should be between 23 and 42 bytes.

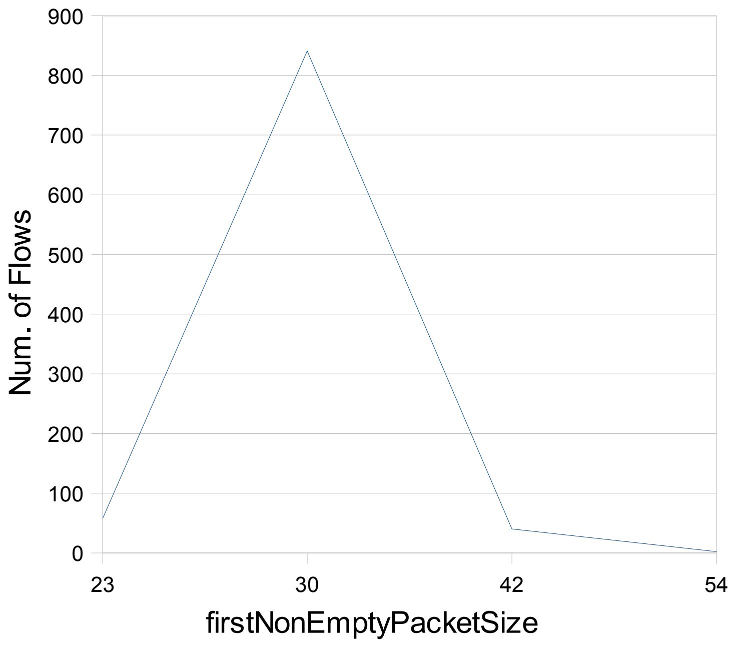


Figure 5.18. firstNonEmptyPacketSize of the Whole Flow

In addition to the above attribute values some other values are also taken into consideration. FirstFewNonEmptyPacketDirections value is observed and seen that it has two characteristics. First there direction of the flows packet is either (1,0,0) that is all three packets are from originator to destination, though the second characteristic is (1,-1,1) that is the first and the third packet is from originator to responder and the second one from responder to originator. The reason of the two different characteristics is that BTP has two different protocol work inside. One is the THP, that will allow the client to contact other peers must be periodically requested from the tracker. The other one is the PWP that is responsible from download.

Transaction value is also observed and it is seen that $\text{ALPHA}_{\text{transaction}}$ is either 0 or 1. File transfer indicator values are also observed and seen that ALPHA, BETA, GAMMA values for both Originator and Responder are equal to zero. By using the above defined criteria, torrent traffic is retested for characterization. The result gathered is as follows:

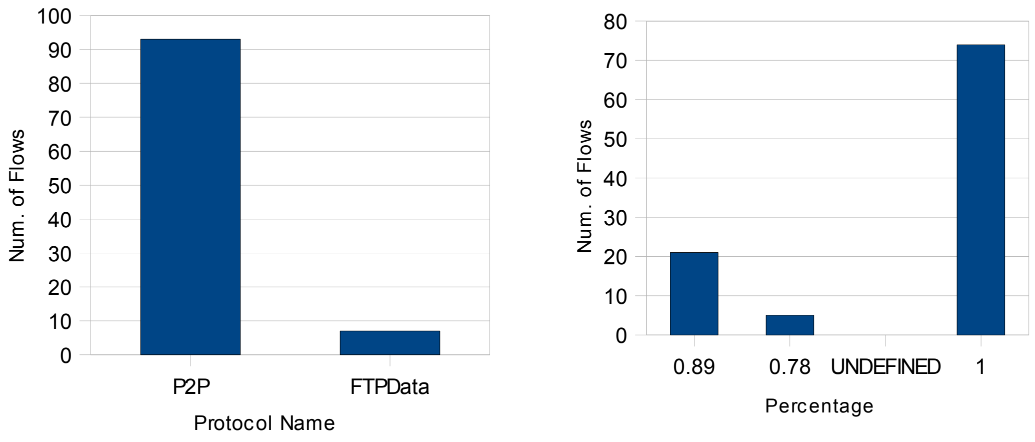
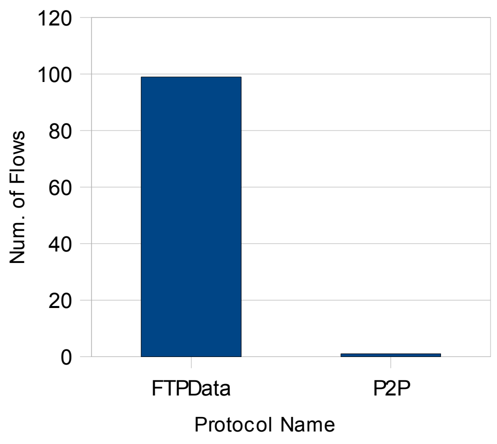


Figure 5.19. P2P Test with FTPData

According to the above figures, 93 flows are detected as BitTorrent protocol, though the rest 7 flows are detected as FTPData. For the recognized peer-to-peer traffic, 21 of them are recognized with 89% percentage, 5 of them are recognized with 78% and 1 of them recognized with 74%.

To be sure from the defined criteria, FTPData, Telnet and HTTP traffic is retested.



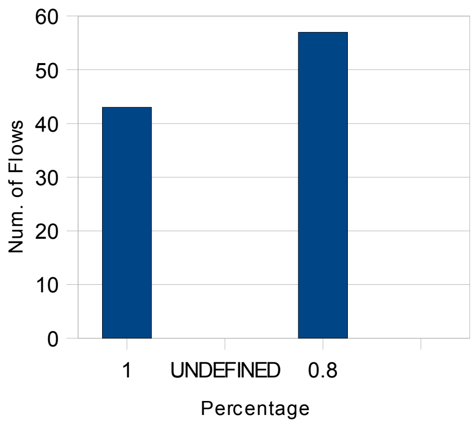
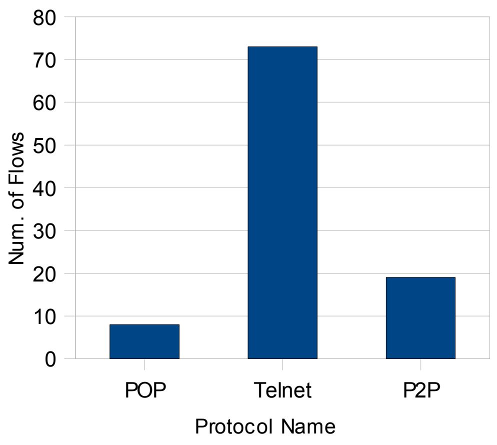


Figure 5.20. FTPData Test with P2P

According to the above graphs, 100 flows related with FTPData is examined. 99 of them are still recognized as FTPData though only 1 of them is determined to be false. For the recognized FTPData, 57 of them are recognized with 80% though the rest 43 of them are recognized with 100%.

For the Telnet traffic, the results are as follows:



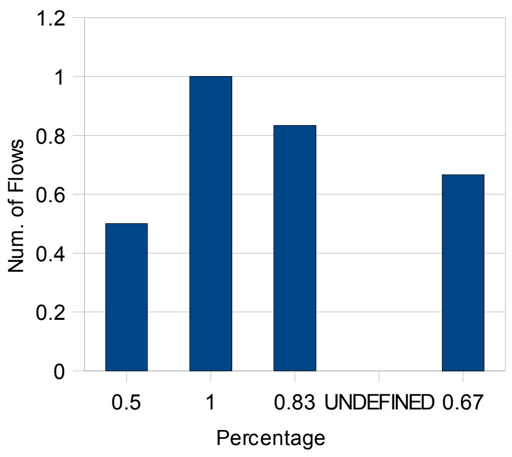


Figure 5.21. Telnet Test with P2P

According to the Telnet test results, 100 flows related with Telnet is examined. 73 of them are still recognized as Telnet, thogh 27 of them is determined as false. For the recognized Telnet flows, 4 of them are recognized with 50%, 43 of them are recognized with 83%, 26 of them are recognized with 67% and the rest 25 of them are recognized with 100%.

The HTTP results are more meaningful that the defined criteria has no effect on HTTP results. The results are as follows.

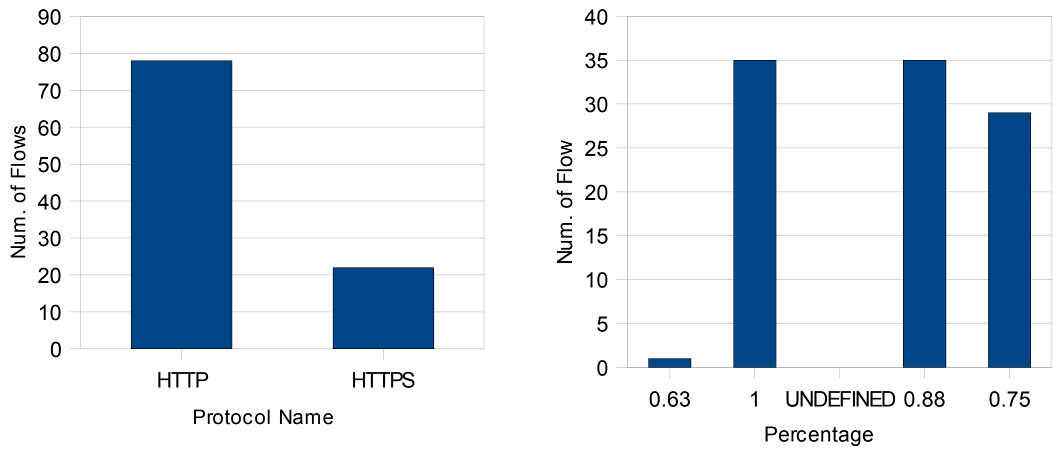


Figure 5.22. HTTP Test with P2P

CHAPTER 6

CONCLUSION

In this thesis it is represented that network traffic can be characterized by using flow characteristics. Flow characteristics are evaluated with a combined method that some metrics are calculated using packet inspection and some requires flow based approach. It is seen that by using the attribute values it is possible to detect many application level protocols. Although calculating the attribute values for each flow takes long time in some cases, they give trustful information that is independent from port numbers. The attributes are easy to calculate that there is no need to follow state information for flow packets, instead easy packet and flow metrics are used to calculate attribute values.

Attribute detection of this thesis is working an autonomous way so it can be used to characterize any application level protocol which is not mentioned at the test results. So as a contribution to the related work (De Montigny and Leboeuf 2005), BitTorrent traffic characteristics are defined. The derived characteristics depend on determination of attributes by observing similarities and patterns. Though, the anomaly detection part is not being handled by a single program, because of the usage of different tools. By using the principle component analysis, it is represented that splitting the undefined traffic eases the anomaly detection. After calculating the number of bytes, packets of IP flow values, applying the PCA causes a graph that has peeks inside which represents the anomalies at the traffic.

Although it is possible to detect byte or packet based anomalies, it is hard to decide what type of anomaly the peek belongs to. This requires an additional inspection of the packets that belong to the suspicious traffic either manually or by using an IDS like snort.

Current solutions to the network anomaly detection depends on either event based or signature based approaches. Both of them require to define the abnormal behavior or traffic either as a signature or as an event expression. This thesis concentrates on the defining the normal traffic instead of suspicious traffic. Suspicious traffic is separated from the whole observed traffic during the characterization period. The rest undefined traffic is marked as for further evaluation. After applying PCA on the undefined traffic, anomalies occurred in a time intervals are detected.

This thesis applies both characterization and anomaly detection mechanism to the network traffic, so it presents a different and applicable approach to the current anomaly detection problem.

This thesis combines packet based and flow based evaluation and represents the method for anomaly detection. Calculating flow attributes is applied for some application level protocols but some are still missing. As a further work additional application level protocols should be taken into consideration, like peer to peer traffic except from BitTorrent and audio/video streams.

Another main further aim can be optimizing the software as if it will be able to analyze real time traffic. Current work is only done by using off line records. This will require to find the bottlenecks of the software where the attribute calculation is taking much time.

The last contribution to this work will be adding recognizers for the detected anomalies. This will enable the analyzer see the type of the anomaly detected. So enhancing this work as a tool that can be used from its graphical user interface which enable real time traffic anomaly detection will be a remarkable contribution to network anomaly detection.

REFERENCES

Basic Analysis and Security Engine (BASE). http://base.secureideas.net/ (accessed May 25, 2009).

BitTorrent (protocol) http://en.wikipedia.org/wiki/BitTorrent_(protocol) (accessed May 25, 2009).

Boddington M. 2009. Utility for processing pcap files http://www.badpenguin.co.uk/main/content/view/46/1/ (accessed May 25, 2009).

CarnegieMellon Software Engineering Institute 2009. SiLK - Documentation. http://tools.netsa.cert.org/silk/silk_docs.html (accessed May 20, 2009).

Cisco Systems 2009. CISCO IOS Netflow. http://www.cisco.com/web/go/netflow (accessed May 20, 2009).

Claffy K., Braun H-W., Polyzos G. October 1995. A Parameterizable Methodology for Internet Traffic Flow Profiling. *IEEE JSAC* 1481-1494.

Cooperative Association for Internet Data Analysis (CAIDA) 2006. cflowd: Traffic Flow Analysis Tool. http://www.caida.org/tools/measurement/cflowd/ (accessed May 20, 2009).

De Montigny A. and Leboeuf. December 2005. Flow Attributes For Use In Traffic Characterization.

Denning D. 1986. An Intrusion-Detection Model. *IEEE Symposium on Security and Privacy*.

Ettercap. http://ettercap.sourceforge.net/ (accessed May 25, 2009).

Fonseca J., Reza B. 2005 BitTorrent Protocol - BTP/1.0 http://jonas.nitro.dk/bittorrent/bittorrent-rfc.html (accessed May 25, 2009).

Gerçek G. A Flow Based Tool for Network Traffic Characterization. Izmir Institute of Technology, License Thesis.

Hernandez-Campos, F., Nobel, A.B., Smith, F.D., and Jeffay, K. 2005. Understanding Patterns of TCP Connection Usage with Statistical Clustering. *13th IEEE International Symposium*. 27(29) 35-44.

Hollmen J. March 1996. Principle component analysis. http://www.cis.hut.fi/~jhollmen/dippa/node30.html (accessed May 20, 2009).

Karagiannis T., Papagiannaki K. and Faloutsos M. August 2005. BLINC: Multilevel Traffic Classification in the Dark. *ACM SIGCOMM.*

Keys K., Moore D., Koga R., Lagache E., Tesch M., and Claffy K. April 2001. The Architecture of CoralReef: an Internet Traffic Monitoring Software Suite. *PAM2001, A workshop on Passive and Active Measurements.*

Ktorrent http://ktorrent.org/ (accessed May 25, 2009).

Lakhina A., Crovella M., and Dio C. 2004. Diagnosing Network-Wide Traffic Anomalies.

Lakhina A., Crovella M., Crovella M. May 2004. Characterization of Network-Wide Anomalies in Traffic Flows.

Lawrence Berkeley National Laboratory 2009. Bro Intrusion Detection System – Bro Overview. http://www.bro-ids.org/ (accessed May 15, 2009).

Lee C.S. Jume 2008. Network Flow: Uni-Directional VS Bi-Directional. http://geek00l.blogspot.com/2008/01/network-flow-uni-directional-vs-bi.html (accessed May 15, 2009).

Lee C.S. May 2008. Training: Practical Network Flow Analysis http://geek00l.blogspot.com/2008/05/training-practical-network-flow.html (accessed May 15, 2009).

Lincoln Labarotary Massachusetts Institute of Technology 2009. 1999 DARPA Intrusion Detection Evaluation Data Set Overview. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html (accessed May 25, 2009).

Luis MG. 2009. TCPDUMP/LIBPCAP public repository. http://www.tcpdump.org/ (accessed May 20, 2009).

Netflow. http://en.wikipedia.org/wiki/Netflow (accessed May 20, 2009).

Nmap - Free Security Scanner for Networ Exploration & Security Audits. http://nmap.org/ (accessed May 25, 2009).

Network Working Group 1999. Traffic Flow Measurement Architecture. CRC Technical Note No. CRC-TN-2005-003 http://www.faqs.org/rfcs/rfc2722.html (accessed May 20, 2009).

Packet (information technology). http://en.wikipedia.org/wiki/Network_packet (accessed May 20, 2009).

Principle component analysis. http://en.wikipedia.org/wiki/Principal_component_analysis (accessed May 20, 2009).

Python Software Foundation 2009. Python Programming Language - Official Web Site. http://www.python.org/ (accessed May 25, 2009).

QoSient, LLC. 2009. ARGUS - Auditing Network Activity. http://qosient.com/argus/ (accessed May 20, 2009).

Realtime Traffic Flow Measurement Working Group (RTFM) December 1995. http://www.ietf.org/proceedings/95dec/ops/rtfm.html (accessed May 20, 2009).

Scapy. http://www.secdev.org/projects/scapy/ (accessed May 25, 2009).

Sourcefire May 2009. What is Snort?. http://www.snort.org/ (accessed May 25, 2009).

Turner A. 2009. Tcpreplay. http://tcpreplay.synfin.net/trac/ (accessed May 20, 2009).

Wireshark. http://en.wikipedia.org/wiki/Wireshark (accessed May 20, 2009).

Xu K., Zhang Z., and Bhattacharya S. 2005. Profiling Internet Backbone Traffic: Behavior Models and Applications. *SIGCOMM.*

Zhang Y. and Paxson V. August 2000. Detecting Backdoors. *USENIX Security Symposium.*

APPENDIX A

Table A.1. Attributes measured over the whole flow

Attributes	**Description**	**Inspection Method**
KEY	A 5-tuple indicating the Originator IP address, the Responder IP address, the IP Protocol (i.e. TCP or UDP), the source port of the Originator, and the source port of the Responder.	Flow Based
BEGIN_TIME	Arrival time of the 1st packet as provided by libpcap.	Flow Based
END_TIME	Arrival time of the last packet as provided by libpcap.	Flow Based
DURATION	Completion time of the flow in microseconds.	Flow Based
FIRST_NONEMPTY_PACKET_SIZE	Payload length of the first non-empty packet.	Packet Based
FIRST_FEW_NONEMPTY_PACKET _DIRECTIONS	An array of 10 discrete values for the directions (-1 or 1) of the first 10 non-empty packets. 1: Originator to Responder, -1: Responder to Originator Array is initialized with values equal to 0 in case fewer than 10 packets contain data.	Packet Based
DATA_BYTE_RATIO_ORIG_TO_RE SP	Total amount of payload data transmitted by the Originator over the Total amount of payload data transmitted by the Responder (initialized to -1 for flows with no data transmitted by the Responder).	Flow Based

(cont. on next page)

Table A.1. (cont.) Attributes measured over the whole flow

Attributes	**Description**	**Inspection Method**
INTERARRIVAL_DISTRIBUTION	A discrete distribution of inter-packet delays represented by an array of 9 continuous binned values. The value in each bin is between 0 and 1 and represents the relative proportion of packets that fell into that bin.	Packet Based
Conversational Indicator		
$\text{ALPHA}_{\text{conversation}}$	The number of non-empty packets that belong to a conversation over the total of nonempty packets.	Packet Based
$\text{BETA}_{\text{conversation}}$	The number of non-empty packets that belong to a sustained conversation over the total of non-empty packets that belong to a conversation.	Packet Based
$\text{GAMMA}_{\text{conversation}}$	The proportion of conversation packets that are transmitted by the originator.	Packet Based
Transaction Indicator		
$\text{ALPHA}_{\text{transaction}}$	Indicator of how often “ping pong” exchanges are seen in a flow.	Packet Based

Table A.2. Attributes measured for each direction of the flow

Attributes	**Description**	**Inspection Method**
INTERARRIVAL_DISTRIBUTION	A discrete distribution represented by an array of 9 continuous values. The array contains the binned values for inter-packet delays in the considered direction. The value in each bin is between 0 and 1 and represents the relative proportion of packets that fell into that bin.	Packet Based
PAYLOAD_DISTRIBUTION	A discrete distribution of packet payload length represented by an array of 23 continuous values. The array contains the binned values for payload lengths per packet.	Packet Based
BYTE_COUNT	Total amount of byte transferred (including bytes found in the network and transport headers).	Flow Based
DATA_BYTE_COUNT	Total amount of byte transferred as payload.	Flow Based
PACKET_COUNT	Total number of packets.	Flow Based
DATAPACKET_COUNT	Total number of non-empty packets.	Flow Based
Encryption Indicators		
$\text{ALPHA}_{\text{chipherblock}}$	Estimated popular GCD among the packet payload lengths	Packet Based
$\text{BETA}_{\text{chipherblock}}$	Ratio of non-empty packet-payload lengths that are divisible by $\text{ALPHA}_{\text{cipherblock}}$	Packet Based
Keystroke Interactive Indicator		
$\text{ALPHA}_{\text{key\_interactive}}$	Indicator of interactive inter-packet departure (for keystroke packets)	Packet Based
$\text{BETA}_{\text{key\_interactive}}$	Indicator of interactivity based on the proportion of small packets	Packet Based
$\text{GAMMA}_{\text{key\_interactive}}$	Indicator of consecutive small packet	Packet Based
$\text{DELTA}_{\text{key\_interactive}}$	Indicator of consecutive small packet	Packet Based
$\text{EPSILON}_{\text{key\_interactive}}$	Indicator of irregularity between inter-arrival of consecutive small packets	Packet Based
Command-line Interactive Indicator		
$\text{ALPHA}_{\text{cmd\_interactive}}$	Indicator of interactive inter-packet departure (for command-line packets	Packet Based
$\text{BETA}_{\text{cmd\_interactive}}$	Indicator of interactivity based on the proportion of small packets	Packet Based
$\text{GAMMA}_{\text{cmd\_interactive}}$	Indicator of consecutive small packets	Packet Based
$\text{DELTA}_{\text{cmd\_interactive}}$	Indicator of piggyback packing	Packet Based
$\text{EPSILON}_{\text{cmd\_interactive}}$	Indicator of irregularity between inter-arrival of consecutive small packets	Packet Based

(cont. on next page)

Table A.2. (cont.) Attributes measured for each direction of the flow

Attributes	**Description**	**Inspection Method**
File transfer Indicators		
$\text{ALPHA}_{\text{file}}$	Indicator of inter-packet departure during a file transfer	
$\text{BETA}_{\text{file}}$	Indicator of file transfer based on the proportion of big packets	Packet Based
$\text{GAMMA}_{\textbf{file}}$	Indicator of consecutive big packet	
$\text{ALPHA}_{\text{constantpacketrate}}$	Indicator of how close to the mean the 5-second packet rate measurements are	Packet Based

APPENDIX B

The appendix contains the rule sets currently used in the profiles of the protocols mentioned at the *De Montigny* and *Leboeuf's*9 work. To satisfy a profile, all of the specified tests must succeed.

1) HTTP web browsing

Test_duration:

Duration > 50000 μsec

Test_transmissionrate:

Originator.$\alpha_{constantbitrate}$ <0.5 && Originator.$\alpha_{constantpacketrate}$ <0.5 &&
Responder.$\alpha_{constantbitrate}$ <0.5 && Responder.$\alpha_{constantpacketrate}$ <0.5

i.e. The transmission rate is more irregular than regular.

Test_payload:

Originator.PayloadDistribution([0-1[) + Originator.PayloadDistribution([180-650[)>0.8 &&
Originator.PayloadDistribution([1-100[)+Originator.PayloadDistribution([1380-inf[)==0

Test_databyteratio:

0.005<DatabyteRatioOrigToResp<4

Test_requestdatabyte:

Originator.DatabyteCount < 21000

Test_firstnonemptypacketsize:

120 < FirstNonEmptyPacketSize < 1000

i.e. The first non-empty packet of the session, which is a HTTP GET, contains at least 120 bytes of data (small URL and only essential HTTP fields) and at most 1000 bytes of data (long URL and many HTTP fields).

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1:2)=[1, -1]

i.e. The first non-empty packet is sent by the Originator (client) and the second is sent by the Responder (server).

Test_noconsecutivesmallpackets:

Originator.$\gamma_{key\_interactive} \leq 0$ && Originator.$\gamma_{cmd\_interactive} \leq 0$ &&
Responder.$\gamma_{key\_interactive} \leq 0$ && Responder.$\gamma_{cmd\_interactive} \leq 0$

2) IMAP

Test_duration:

Duration > 100000 µsec

Test_transmissionrate:

Originator.$\alpha_{constantbitrate}$ <0.5 && Originator.$\alpha_{constantpacketrate}$ <0.5 &&
Responder.$\alpha_{constantbitrate}$ <0.5 && Responder.$\alpha_{constantpacketrate}$ <0.5

i.e. The transmission rate is more irregular than regular.

Test_payload:

Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([5-180[)>0.8
&&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([20-100[)>0.2

Test_databyteratio:

DatabyteRatioOrigToResp<1

i.e. The server sends more data than the client.

Test_firstnonemptypacketsize:

10 < FirstNonEmptyPacketSize < 250

i.e. The first non-empty packet, which is sent by the mail server, is typically small ("OK" + optional info such as server version, name, capabilities, etc.).

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1:5)=[-1, 1,-1, 1,-1] ||
FirstFewNonEmptyPacketDirections(1:6)=[-1,-1, 1,-1, 1,-1]

i.e.

1) Responder describes server,

2) (optional) If client remains quiet, server sends an empty response. From this point, IMAP behaves like a Request/Response protocol driven by the client Requests. For instance, the following sequence may follow:

3) Originator asks for capability

4) Responder responds with capability

5) Originator sends login & password

6) Responder accepts/rejects login

Test_nonemptypacketratio:

Originator.datapacketcount/Originator.packetcount > 0.5 &&
Responder.datapacketcount/Responder.packetcount > 0.6

i.e At least 50% of the packets sent by the client carry data, and at least 60% of the packets sent by the mail server carry data.

3) POP

Test_duration:

100000 < Duration < 10000000 µsec

i.e. In contrast with IMAP, POP terminates the session once mail messages have been downloaded, this typically takes less than 5 seconds (say a maximum of 10 seconds to set a loose threshold).

Test_transmissionrate:

Originator.$\alpha_{constantbitrate}$ <0.5 && Originator.$\alpha_{constantpacketrate}$ <0.5 &&
Responder.$\alpha_{constantbitrate}$ <0.5 && Responder.$\alpha_{constantpacketrate}$ <0.5

i.e. The transmission rate is more irregular than regular.

Test_payload:

Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([5-20[)>0.9 &&
Originator.PayloadDistribution([1-5[)==0 &&
(Responder.PayloadDistribution([0-50[)+Responder.PayloadDistribution([236-269[)+
Responder.PayloadDistribution([516-549[)+Responder.PayloadDistribution([1432-1473[))>0.6

Test_databyteratio:

DatabyteRatioOrigToResp<0.65

i.e. The server sends more data than the client.

Test_firstnonemptypacketsize:

10 < FirstNonEmptyPacketSize < 100

i.e. The first non-empty packet, which is sent by the mail server, is typically small ("OK" + optional info such as server version, and name). POP server responses tend perhaps to be smaller than IMAP server responses.

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1:5)=[-1, 1,-1, 1,-1] ||
FirstFewNonEmptyPacketDirections(1:6)=[-1,-1, 1,-1, 1,-1]

i.e. Similar to IMAP with regards to the initial directional dynamics.

Test_nonemptypacketratio:

Originator.datapacketcount/Originator.packetcount < 0.7 &&
Responder.datapacketcount/Responder.packetcount > 0.5

i.e At most 70% of the packets sent by the client carry data, and at least 50% of the packets sent by the mail server carry data.

4) SMTP

Test_duration:

100000 < Duration < 10000000 µsec

i.e. SMTP terminates the session once mail messages have been transferred, this typically takes less than 5 seconds (say a maximum of 10 seconds to set a loose threshold).

Test_transmissionrate:

Originator.αconstantbitrate <0.5 && Originator.αconstantpacketrate <0.5 &&
Responder.αconstantbitrate <0.5 && Responder.αconstantpacketrate <0.5

i.e. The transmission rate is more irregular than regular.

Test_payload:

(Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([5-100[)+
Originator.PayloadDistribution([236-269[)+ Originator.PayloadDistribution([516-549[)+
Originator.PayloadDistribution([1432-1473[))>0.6 &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([5-100[)>0.8 &&
Responder.PayloadDistribution([1-5[)+Responder.PayloadDistribution([350-inf[)==0

Test_databyteratio:

DatabyteRatioOrigToResp > 1

i.e. The databyte ratio Originator To Responder is greater than one, typically MUCH greater than 1.

Test_firstnonemptypacketsize:

20 < FirstNonEmptyPacketSize < 300

i.e. SMTP server responses are typically around a hundred bytes. We chose loose boundaries (20 and 300 bytes).

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1:5)=[-1, 1,-1, 1,-1]

i.e.

1) Responder describes server. From this point, SMTP behaves like a Request/Response protocol driven by the client Requests. For instance, the following sequence may follow:

2) Originator sends Client Helo

3) Responder sends Server Helo

4) Originator sends AUTH command

5) Responder accepts/rejects

Test_nonemptypacketratio:

Originator.datapacketcount/Originator.packetcount > 0.5
i.e At least 50% of the packets sent by the client carry data.

Test_datapacketcount:

4 < Responder.datapacketcount < 15
SMTP servers respond with a somewhat fixed number of non-empty packets.

Test_databytecount:

300 < Responder.databytecount < 900
SMTP servers respond with a somewhat fixed number of data bytes.

5) SSH

Test_payload:

Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([10-180[) > 0.8 &&
Originator.PayloadDistribution([1-10[)==0% && Responder.PayloadDistribution([1-10[)==0 &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([10-180[) > 0.5

Test_databyteratio:

DatabyteRatioOrigToResp<1
i.e. The server sends more data than the client.

Test_cipherblock:

mod(Originator.$\alpha_{cipherblock}$, 4)==0 && mod(Responder.$\alpha_{cipherblock}$, 4)==0 &&
Originator. $\beta_{cipherblock} > 0.8$ && Responder. $\beta_{cipherblock} > 0.8$
i.e. At least 80% of the non-empty packets must be divisible by 4.

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1)= -1
i.e. The first non-empty packet is sent by the Responder (server).

Test_nonemptypacketratio:

Responder.datapacketcount/Responder.packetcount > 0.5

i.e At least 50% of the packets sent by the server carry data.

6) TELNET

Test_payload:

Originator.PayloadDistribution([0-10[) > 0.8 && Originator.PayloadDistribution([350-inf[)==0

Test_databyteratio:

DatabyteRatioOrigToResp<0.2 && Originator.datapacketcount < Responder.datapacketcount

i.e. The server sends much more data than the client. The server also sends more non-empty packets.

Test_firstnonemptypacketsize:

FirstNonEmptyPacketSize < 30

i.e. empirical estimate based on a max of 10 options negotiated.

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1:2)= [-1, 1]

i.e. The first non-empty packet is sent by the Responder (server) and the second is sent by the Originator.

Test_nonemptypacketratio:

Responder.datapacketcount/Responder.packetcount > 0.4

i.e At least 40% of the packets sent by the server carry data.

Test_transaction:

$\alpha_{transaction} > 0.7$

i.e Telnet is mostly transactional.

7) FTPCommand

Test_duration:

Duration > 500000 µsec

Test_payload:

Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([5-100[)>0.8 &&
Originator.PayloadDistribution([1-5[)+Originator.PayloadDistribution([350-inf[)==0 &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([10-180[)>0.8 &&
(Responder.PayloadDistribution([1-5[)+Responder.PayloadDistribution([350-650[)+
Responder.PayloadDistribution([1000-inf[))==0

Note that while the responder also avoids sending big packets, it was not unusual to see packets containing between 650 and 1000 bytes of payload, in particular when transmitting “code 220” for greetings and warnings.

Test_databyteratio:

0.1 < DatabyteRatioOrigToResp < 0.5 && Originator.datapacketcount < Responder.datapacketcount

i.e. The server sends more data and non-empty packets than the client.

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1)= -1

i.e. The first non-empty packet is sent by the Responder.

Test_nonemptypacketratio:

Responder.datapacketcount/Responder.packetcount > 0.6

i.e. At least 60% of the ftpcmd packets sent by the server carry data.

Test_transaction:

αtransaction > 0.95

i.e. FTPcommand is mostly (if not completely) transactional.

Test_noconsecutivebigpackets:

Responder.γfile ≤ 0

i.e While IMAP and FTPcommand are similar with respect to the other criteria, ftp server tend not to transmit consecutive big packets in the FTPcommand connection.

8) FTPDATA

Test_payload:

(Originator.PayloadDistribution([1-5[)==0 && Responder.PayloadDistribution([0-1[)==1)
||
(Responder.PayloadDistribution([1-5[)==0 && Originator.PayloadDistribution([0-1[)==1)

Test_databyteratio:

DatabyteRatioOrigToResp == 0 || DatabyteRatioOrigToResp == -1

i.e. The data is flowing in one direction only. The value is -1 if the transmitting end is the Originator, and the value is 0 if the Responder is the transmitting end. A value of -1 can be associated to two cases: the transfer is an ACTIVE get or a PASSIVE put, depending on whether the Originator is the FTP server or the FTP client respectively. Similarly, a value of 0 indicates an ACTIVE put or a PASSIVE get, depending on whether the Originator is the FTP server or the client respectively. The role of the Originator can be determined by examining related flows marked as FTPcommand.

Test_databytecount:

Originator.DatabyteCount + Responder.DatabyteCount > 0

i.e. As a rule of thumb, a FTPdata session involves transferring data... therefore there should be packets carrying data in at least one of the direction.

Test_packetcount:

0.3 < (Originator.PacketCount/(Originator.PacketCount+Responder.PacketCount)) < 0.7

i.e. The amount of packets transmitted in each direction is similar.

Test_nonemptypacketratio:

(Originator.datapacketcount==0 && Responder.datapacketcount/Responder.packetcount > 0.3) ||
(Responder.datapacketcount==0 && Originator.datapacketcount/Originator.packetcount > 0.3)

This rule typically holds provided there are more than 5 packets in each direction.

9) HTTPS

Test_duration:

Duration > 50000 µsec

Test_transmissionrate:

Originator.αconstantbitrate <0.5 && Originator.αconstantpacketrate <0.5 &&
Responder.αconstantbitrate <0.5 && Responder.αconstantpacketrate <0.5

i.e. The transmission rate is more irregular than regular.

Test_payload:

Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([50-180[) > 0.6 &&
Originator.PayloadDistribution([1-5[)+Originator.PayloadDistribution([1000-inf[)==0 &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([20-100[)+
Responder.PayloadDistribution([549-inf[) > 0.6

Test_datapacketcount:

Originator.datapacketcount<10

Test_firstnonemptypacketsize:

90 < FirstNonEmptyPacketSize < 250

i.e. The first non-empty packet of direct SSL connections (a SSL Client Helo packet) is typically small (contains very few cipher specifications).

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1:2)=[1, -1]

i.e. The first non-empty packet is sent by the Originator (client) and the second is sent by the Responder (server).

Test_conversation:

(Originator.αconversation > 0.25 && Originator.datapacketcount ≥ 5) ||
Originator.datapacketcount < 5

10) RLOGIN

Test_payload:

Originator.PayloadDistribution([0-5[) > 0.8 &&
Originator.PayloadDistribution([350-inf[)==0

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1)= 1

i.e. The first non-empty packet is sent by the Originator.

Test_nonemptypacketratio:

Responder.datapacketcount/Responder.packetcount > 0.4

i.e. At least 40% of the packets sent by the server carry data.

Test_conversation:

αconversation > 0.01 && βconversation > 0.4 && γconversation > 0.6

i.e. RLOGIN appears a little like a conversation (compared to SSH,TELNET, and FTPcommand). When conversing, the Originator sends more packets than the Responder.

11) MSNChat

Test_payload:

Originator.PayloadDistribution([0-1[) + Originator.PayloadDistribution([100-450[) > 0.8
&&
Originator.PayloadDistribution([1-5[)==0 && Responder.PayloadDistribution([1-5[)==0 &&
Responder.PayloadDistribution([0-1[) + Responder.PayloadDistribution([100-450[) > 0.8

Test_databyteratio:

0.1 < DatabyteRatioOrigToResp < 10

i.e. This assumes that one of the user may be at most 10 times chattier than the other.

Test_firstnonemptypacketdirections:

FirstFewNonEmptyPacketDirections(1)= 1

i.e. The first non-empty packet is sent by the Originator.

Test_interactive:

Originator.αcmd_interactive > 0.3 && Originator. βcmd_interactive > 0.6 &&
Originator.γcmd_interactive > 0.6 &&

Originator.δcmd_interactive > 0.3 && Originator.εcmd_interactive > 0.3 &&
Responder.αcmd_interactive > 0.3 && Responder. βcmd_interactive > 0.6 &&
Responder.γcmd_interactive > 0.6 &&
Responder.δcmd_interactive > 0.3 && Responder.εcmd_interactive > 0.3

i.e. Each direction is command-line interactive.

Test_conversation:

αconversation > 0.4 && βconversation > 0.4 && (0.35 < γconversation < 0.65)

i.e the flow must have conversational episodes (αconversation), it must have sustained conversation episodes (βconversation) and the amount of packets belonging to a conversation must be similar in each direction (γconversation).