Identification of Hostile TCP Traffic using Support Vector Machines

Glenn Wilkinson
Lincoln College
Oxford University Computing Laboratory

Supervised by: Dr Andrew Ker

A thesis submitted in partial fulfillment for the degree of
Master of Science

Trinity 2009

Abstract

Our research involved the application of supervised learning techniques to the field of network based intrusion detection systems. We developed a framework for extracting various feature sets from offline network traffic datasets. The particular dataset we used was the DARPA intrusion detection off-line evaluation set which contain raw network dump files along with flat files containing labels indicating attack traffic. We made use of Support Vector Machines to classify the processed and labeled network traffic into "attack" and "normal" sets, as well as multiclass classification in which traffic was classified into various attack categories.

We focused only on analyzing TCP traffic and our feature sets were representative of entire TCP sessions, as opposed to individual packets. The feature set of most interest to us was that generated from the content of TCP flows. The majority of intrusion detection systems involving learning techniques discard traffic payload altogether. We demonstrated the potential use of inspecting the content of TCP streams with simple analysis techniques.

"As a young boy, I was taught in high school that hacking was cool."
-Kevin Mitnick

Contents

List of Tables

List of Figures

List of Frequent Acronyms

CCI	Correctly Classified Instances
DARPA	Defense Advanced Research Project Agency
DoS	Denial of Service
DR	Detection Rate
FNR	False Negative Rate
FPR	False Positive Rate
IDS	Intrusion Detection System
IP	Internet Protocol
ML	Machine Learning
MSVM	Multiclass Support Vector Machine
OSI	Open System Interconnection
QP	Quadratic Programming
ROC	Receiver Operating Characteristic
SMO	Sequential Minimal Optimization
SOM	Self Organising Map
SVM	Support Vector Machine
TCP	Transmission Control Protocol

Acknowledgments

To my supervisor Andrew, thank you for your guidance and inspiration during this project. You have taught me the principles of being a good scientist. Thank you to my friends and colleagues at SensePost Ltd in South Africa, for igniting my passion for the field of information security. Thank you to Abi, Ashley and especially Anna for your hours of proof reading. Thank you to my family in Zimbabwe, who have supported me all the way.

Finally, I acknowledge the funding generously provided to me throughout my studies in Oxford by the Rhodes Trust.

Chapter 1

Introduction

In this research paper we explore various characteristics of network traffic which can be used to discern hostile Transmission Control Protocol (TCP) sessions from normal ones with machine learning (ML) techniques. Simple metrics regarding the payload content of TCP traffic were of most interest to us, but we also calculated features from the characteristics of packet headers, inter-packet relations and interflow relations. It was not our intention to build a real-time intrusion detection system (IDS), but rather to investigate the utility of these features by analysing offline traffic dumps. Classification was done using the Weka framework with the Support Vector Machine (SVM) algorithm. We did not implement our own algorithm, or tune Weka's SVM, but rather we treated it as a "black box".

In this chapter we compare various approaches to intrusion detection and examine other work related to our research. In Chapter 2 we describe the SVM algorithm and relate it to the context of intrusion detection. Chapter 3 is concerned with data preprocessing and feature extraction- we examine the Defense Advanced Research Project Agency (DARPA) training sets, highlight their limitations and examine possible alternatives. We go on to explain how the network data was preprocessed and describe the feature extraction process (via a set of tools we developed in the Perl programming language). In Chapter 4 we describe how the extracted features were imported to the Weka framework and explain the classification experiments we conducted. Chapter 5 presents the results to these experiments and offers a discussion of our findings. We then present a conclusion and ideas for future work in Chapter 6.

In the appendices we list SQL table definitions, source code and other information should the reader wish to look at our data in more detail.

1.1 Background

The concept of *intrusion detection* was first put forward by J.P Anderson in 1980 [8]. The goal of an IDS is to detect attempts at violating security paradigms of some system [31]. Two main classes of IDS exist: network based and host based. A host based IDS focuses on monitoring system and application software behaviour on an individual computer (for example, following system calls, privilege escalations, etc.). On the other hand, a network based IDS is deployed in a network in

such a position so as to analyse network traffic between hosts. Its goal is to detect traffic which may be indicative of an attack. Our research involved network based IDSs, and any reference to an intrusion detection system in the rest of the paper refers specifically to this class.

The majority of IDSs are based on individual packet inspection coupled with either header features or simple signature matching in payloads. In recent years the threat of zero day attacks[1], the migration of network based attacks to the application layer, and IDS evasion techniques have prompted research into more sophisticated systems. Whilst the idea of applying ML techniques to intrusion detection is not new (albeit primarily in academia as opposed to commercial systems), the examination of the payload is fairly novel- the majority of ML approaches discard the payload and focus on finding patterns in the packet headers. The primary reason for discarding the payload is due to the *curse of dimensionality*: the complexity of learning algorithms scales up drastically with an increase in the number of features to consider whilst the detection rates decrease correspondingly [31]. Representing the many thousands of bytes in a single TCP stream would be infeasible. One solution to this representation would be to apply dimension scaling algorithms or principle component analysis. We instead calculate a few simple metrics over the data extracted from the TCP flow.

1.2 Research Direction

Our primary research goal was to investigate the potential use of simple TCP content stream analysis in the field of intrusion detection. Examining nearly 3 million TCP sessions, we extracted bidirectional payload features such as unigram and bigram entropy, counts of frequent characters and other simple metrics. In addition to payload features, we built up three other feature sets from our data: statistical features from all packet headers, statistics regarding interpacket relations within each session (such as packet interarrival times) and interflow statistics (building features concerning the relationship a TCP session has with temporal *near* sessions, such as counting the number of times the two hosts communicated in the last 10 seconds).

We applied the SVM algorithm to extracted features in order to classify network traffic as hostile or benign. We made use of the Weka framework and the Sequential Minimal Optimization (SMO) implementation of the SVM algorithm to accomplish this. We evaluated our features using a labeled dataset of raw network traffic.

1.3 Supervised vs Unsupervised Machine Learning

ML has been defined as an area of artificial intelligence (A.I.) relating to the development of techniques allowing computers to *learn,* or more specifically, to the creation of algorithms whose performance can grow over time. The field is linked to statistics in the sense that both fields analyse data but, unlike statistics, ML is concerned with the "creation of algorithms of tractable computational complexity"[31].

Within the field of ML there exist two broad learning approaches; supervised and unsupervised. Supervised algorithms create models from labeled datasets (i.e. a set of data in which the desired

[1]Unpublished or undisclosed attacks

Table 1.1: Comparison Between Misuse and Anomaly IDS

Misuse Based (Supervised)	Anomaly Based (Unsupervised)
Requires large labeled datasets for training	Can be trained on unlabeled datasets
Requires updates	Does not require updates
Difficulty in detecting new attacks	Good performance in detecting new attacks
Low false positive rate	High false positive rate
Design is not complicated	Difficult to design
Precise alerts	Vague alerts

outputs from inputs is present). Given enough samples of desired input to output examples the algorithm can predict which category a new sample belongs in. The case where the labels (or desired outputs) are discrete is known as *classification*. Unsupervised algorithms attempt to create a model from *unlabeled* datasets. The approach is closely related to *density estimation* in the field of statistics and attempts to find patterns in the supplied data [31]. A common approach in unsupervised learning is that of *clustering*.

Specific examples of supervised learning algorithms include SVMs, decision trees (e.g. C4.5), k-nearest neighbour and multi-layer perceptrons. Examples of unsupervised learning algorithms include k-means clustering, single linkage clustering, Self Organizing Maps (SOMs) and Adaptive Resonance Theory (ART). For our research we focused on *supervised* learning, specifically making use of the SVM algorithm. The algorithm is explored in detail in Chapter 2.

1.4 Intrusion Detection Systems

The application of ML to the field of IDSs has gained popularity in recent years. There are two broad categories of such techniques: misuse (using supervised algorithms) and anomaly detection (using unsupervised algorithms). Anomaly detection systems aim to detect unusual activity patterns in observed data [31]. Misuse systems analyze large data sets containing labeled samples of known attacks: the system learns to recognise attack patterns and can identify the same class of attacks it was trained on, as well as similar ones. More formally, Zanero and Savaresi present the following two definitions [31]:

"A misuse detection IDS uses a knowledge base (often called a set of signatures) in order to recognize directly the intrusion attempts, which means that instead of trying to describe the normal behavior of a system it tries to describe the anomalous behaviors."

"An anomaly detection IDS tries to create a model of normal behavior for the monitored system(s) or for their users, and flags as suspicious any deviation from this "normal" behavior which exceeds carefully tuned thresholds."

Additionally, the authors compare advantages and disadvantages of both approaches. We present a summary of their arguments in Table 1.1.

In the paper *Learning Intrusion Detection: Supervised or Unsupervised?*[25] the authors formally compare supervised and unsupervised approaches. The results of their research indicate that supervised techniques perform significantly better than unsupervised techniques when identifying *known* attacks. On identifying *unknown* attacks (i.e. testing on attacks not observed during training) the

performance of the supervised method drops drastically, and the unsupervised method performs better. The main problem with the *supervised* method is the lack of available labeled data sets on which to train such systems.

The majority of IDSs which make use of learning techniques focus on the header information of the packets and discard the payload. Those that investigate the payload generally do so with a signature based approach where specific strings of known attacks are looked for. The problem with the signature based approach is that of keeping the database of the signatures updated. In the antivirus field this has worked well as it is trivial to get a sample of a spreading virus so as to extract a signature. However, this approach does not lend itself to the intrusion detection field as it is much harder to obtain samples of recent attack methods, especially for undisclosed ('zero day') attacks. Approaches which discard the payload may be unable to detect a large proportion of attacks- most modern attacks can only be detected via the content of the communication, as the header fields match those of normal traffic. The payload is generally discarded due to the difficulty in representing it. The header fields of network traffic are well defined and only take a certain range of values and therefore are easily represented as features in a ML environment. The payload of an IP packet has a variable length and a typical Maximum Transmission Unit (MTU) on the Internet of 1500 bytes. It is unfeasible to use a set of features to directly accommodate this varied length and large size (i.e. the curse of dimensionality).

1.5 Related Work

Significant recent work in the area of content analysis has been conducted by Stefano Zanero whose research involved analysing individual packet payloads using *unsupervised* learning techniques (specifically using SOMs) [37, 31]. His results highlight the usefulness of packet inspection techniques in an unsupervised context.

Several other authors have applied similar unsupervised approaches, but have discarded the payload: Piotr and Heywood implementing a clustering system using 6 TCP connection features from every connection in the DARPA dataset [26]. Labib and Vemuri developed a real time system called NSOM (Network-based detector using SOM) which was useful at detecting denial of service attacks [18].

Moore and Zuev made use of statistical header based information to classify network traffic between categories such as WWW, mail, bulk, peer2peer, games etc [6]. We made use of the same tool (*tcptrace* [24]) to generate our header based statistical features. In another paper, Moore inspects content to classify traffic but using a signature based approach [7].

Lee and Stolfo explored data mining techniques applied to the headers of network traffic [35]. Barbara and co-workers developed a tool called ADAM (Audit Data Analysis and Mining) which uses data mining techniques to creates association rules to detect anomalies in TCP connections [11]. Mahoney and Chan created performed simple statistical modeling to data extracted from headers with PHAD (Packet Header Anomaly Detection) [21].

Chapter 2

Support Vector Machines

For the task of classifying network traffic we made use of the SVM algorithm. In this chapter we describe how the algorithm works but we must note that we made no attempt to design our own implementation. That is to say we made use of Weka's SVM implementation which we treated as a "black box".

2.1 Introduction

In 1979 Vladimir Vapnik invented the Support Vector Machine (SVM) which he described in his 1982 paper *Estimation of Dependences Based on Empirical Data* [34]. SVMs have been applied to a wide range of classification problems over the years. We see examples of their use in recognition of handwritten text, face detection and pedestrian detection. They have been noted as being the best classifier for hand-written character recognition [6]. SVMs are similar to perceptron neural networks; an SVM which uses a sigmoid kernel function is identical to a two layer neural network. We refer the reader to the following articles which detail how SVMs work: [2, 4, 27, 13]. We summarize ideas from these papers and contextualize them to our problem in the rest of this chapter.

2.2 Binary Classification

A SVM performs binary classification, which means it classifies input into one of two categories. For our usage, this would be "attack traffic" or "normal traffic". The SVM performs its classification by building an p-dimensional hyperplane from n attributes (where *p=n-1*) which results in the *best* separation of the data. Within the SVM a single data instance is represented by a vector, where the vector is constructed from the features of the data. For example, if we were to choose the features *source port* and *payload size* we would have a two dimensional vector which could be depicted on an <x,y> Cartesian plane. This can be represented formally where we are given a set of training data as [2]:

$D = \left\{(x_i, c_i)\,|x_i \epsilon R^p, c_i \epsilon \{-1, 1\}\right\}_{i=1}^{n}$

where c_i is either -1 or 1 (attack or not) which indicates the class to which x belongs.

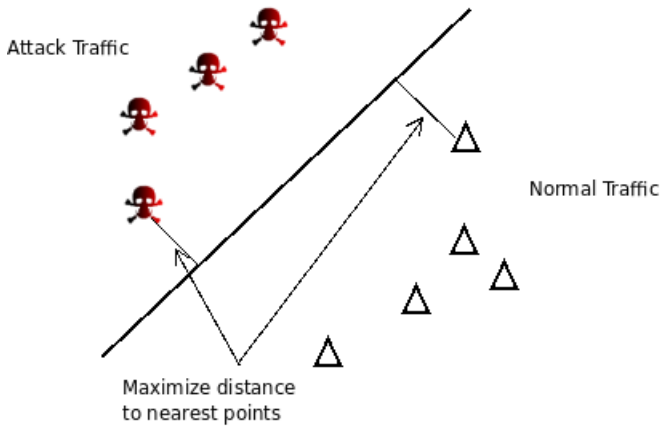


Figure 2.1: Input Space for Linear Support Vector Machine

2.3 Hyperplanes

The goal of the SVM is to separate all the vectors with a hyperplane (in our 2D case, the hyperplane would be a line) such that all instances of one category of the target variable are on one side of the hyperplane, and all instances of the other category on the other side. Our simple two dimensional case is represented in Figure 2.1 (adapted from [27]) where we observe the hyperplane separating positive examples (say, attacks) from negative examples (say, normal traffic) after plotting their 2D vector representations.

There exist an infinite number of possible options for the hyperplane. Therefore we must next consider how we choose the *best*, or most optimal one. The distance between the hyperplane and the nearest positive sample plus the distance between the hyperplane and the nearest negative sample is called the *margin.* The dashed lines in Figure 2.2 (adapted from [13]) parallel to the hyperplane represent this distance. In this figure we observe how the "support" in SVM derives its name- from the vectors nearest the margin hyperplanes which "support" them. The SVM analysis finds a hyperplane such that the margin between these support vectors is maximized.

A hyperplane can be written as the set of points from x which satisfy:

$w \cdot x - b = 0$

where w is perpendicular to the hyperplane and is normalized (as can be seen in Figure 2.2). Therefore $\frac{b}{||w||}$ represents the offset of the hyperplane from the origin along w. In order to obtain the greatest margin we must choose optimum values for w and b to create two supporting hyperplanes as far apart as possible. In the same figure we observe these hyperplanes being described by the formulae:

$w \cdot x - b = 1$

$w \cdot x - b = -1$

In our simple example this is $\frac{2}{||w||}$ meaning we wish to minimize the value of $||w||$. This results in a large quadratic programming (QP) optimization problem of which the exact methods and mathematics do not need to be discussed here but can be found in [27, 34]. John Platt developed

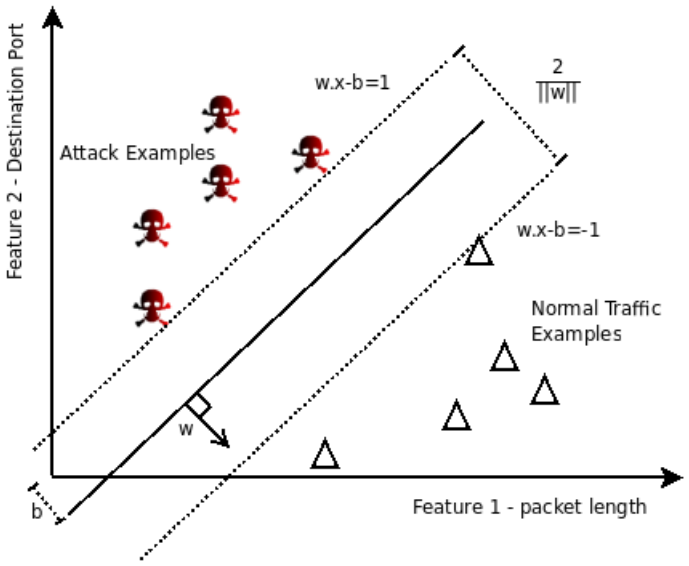


Figure 2.2: Hyperplane with Margin

an algorithm in 1998 for training SVMs called Sequential Minimal Optimization [27]. The SMO breaks down this large QP problem into several smaller QP problems which are solved analytically. The details of the algorithm can be found in Platt's paper, *A Fast Algorithm for Training Support Vector Machines* [27]. Weka has an SMO implementation, which we installed and used for our classification requirements.

2.4 Non-linear Case

Our simple example so far has been based on a linear example- that is to say the two classes of "attack" and "normal" can be separated by a single straight line. This is very unlikely, and so we will now discuss the non-linear case. One option for dealing with this non-linear case would be to fit non-linear curves to the data. However, SVMs handle this via the use of a *kernel function*[20]. This kernel function maps the data into a different space, one in which a hyperplane can separate the data. In the higher dimensional space, the original non-linear observations can now be linearly separated. This is done using Mercer's theorem [20] which states that "a kernel function k(x,y) can be expressed as a dot product in a higher dimensional space." The kernel function transforms formulae which depend on the dot product between two vectors, as is the case here with w and x. The dot product is simply replaced with the kernel function transforming a linear algorithm into a non-linear one. For example, the kernel function could map a 2D set which cannot be separated by a line into a 3D space in which a plane can separate the data. This can then be extended to any n dimensions. This higher dimension mapping allows the SVM to separate data with very complex boundaries.

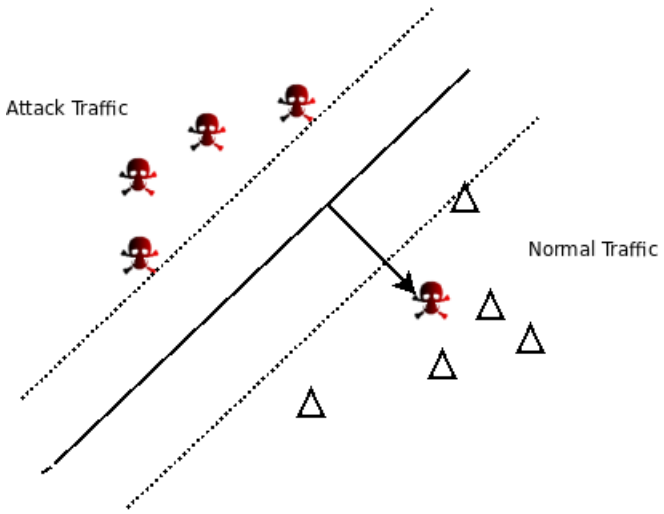


Figure 2.3: Non-separable Training Data

2.5 Non-separable Training Sets

Another problem to consider is when separating all data instances into the classes $c = 1$ or $c = -1$ is not feasible, as demonstrated in Figure 2.3 (adapted from [4]). Linear separation is still used but now training errors are admitted. A cost parameter C is added which adds flexibility to the model by creating a *soft margin* which permits some misclassifications. A greater value of C results in a greater cost of misclassification and results in a model with less generalization.

2.6 Multiclass Support Vector Machines

Binary classification separates data into two classes; in our case "attack" and "normal". We also wanted to run classifications which would separate our data into various attack classes, or the normal traffic class. In order to do this we made use of a Multiclass Support Vector Machine (MSVM). There are several methods for performing multiclass classification with an SVM. The approach Weka takes is to split the multiclass problem into a number of binary problems, such that a binary classifier is constructed for every pair of classes. For our case of DoS, exploit, probing, warez and normal classes we have 10 possible pair combinations (i.e. $\frac{C(C-1)}{2}$, where C is the number of different classes). The binary classifier C_{ij} is trained on examples from the class a_i and the class a_j where samples from a_i are positive and samples from a_i are negative [17]. If classifier C_{ij} classifies a new sample as a_i, then the count for class a_i increases by one; if the sample is classified as a_j then the count for a_j increases by one. The new sample is assigned to the class with the highest score at the end. This is known as the "max-wins" voting strategy. We demonstrate an example of this in Table 2.1 in which a new sample is classified as exploit.

Table 2.1: Multiclass SVM Example

Classifier	Classes		Classified As
C_{dw}	DoS	Warez	Warez
C_{dp}	DoS	Probing	DoS
C_{de}	DoS	Exploit	**Exploit**
C_{dn}	DoS	Normal	DoS
C_{wp}	Warez	Probing	Probing
C_{we}	Warez	Exploit	**Exploit**
C_{wn}	Warez	Normal	Normal
C_{pe}	Probing	Exploit	**Exploit**
C_{pn}	Probing	Normal	Normal
C_{en}	Exploit	Normal	**Exploit**

(Exploit "wins" with 4 votes)

Chapter 3

Data Preprocessing and Feature Extraction

We start this chapter by discussing the DARPA IDS evaluation datasets. We explain the datasets' original purpose and structure as well as their criticisms and shortcomings. We mention possible alternatives to the DARPA dataset and justify why we used it. From here, we move on to the subject of feature extraction. We explain why we chose the features we did, present a set of tools we developed to extract them and describe the process of extracting them.

3.1 DARPA Intrusion Detection Datasets

DARPA is the central research and development organization for the U.S DoD (United States Department of Defense). The organization released an intrusion detection evaluation dataset which in part consisted of network traffic generated from a simulated military base network. The simulated network incorporated hundreds of users utilizing thousands of hosts. The network traffic consisted predominantly of normal traffic but also included a large number of various attacks during the 7 weeks worth of training data and 2 weeks of testing data. Separate flat files are available which list every attack during these 9 weeks. The network topology of the simulated network is depicted in Figure 3.1. The attacks were grouped into one of 4 categories: User to Root; Remote to Local; DoS and Probe. These attacks and their descriptions are summarized in Table 3.1. The attack name *warez* was grouped under the denial of service category. We decided to rather include it in its own group. We combined the attack types of User to Root and Remote to Local into one class as *exploit* due to their similarity and the very small number of user to root attacks present in the dataset.

Of the 7 weeks worth of training data, we chose to make use of weeks 4,5,6 and 7 and made use of both weeks of testing data. We performed a variety of experiments; some in which the 6 weeks of data were combined and classification was performed using cross fold validation, some in which the 6 weeks were combined but we trained and tested on separate attack classes, and finally some in which we trained on the official training data and tested on the official testing data.

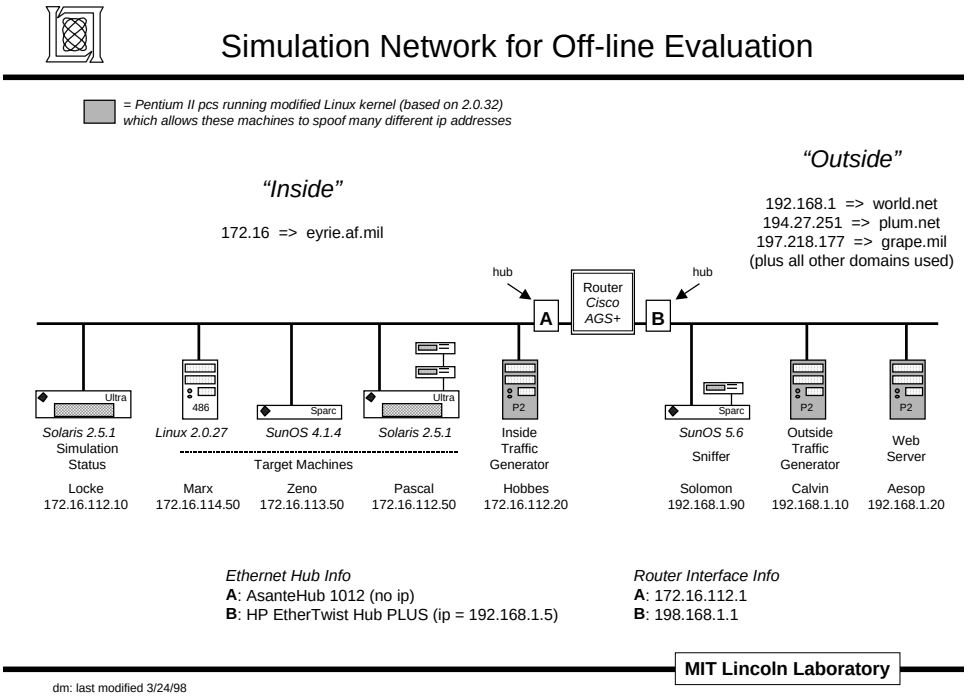


Figure 3.1: DARPA Simulated Network [3]

Table 3.1: DARPA Attacks [3]

Attack Type	Description
Denial of Service (DoS)	Attacker tries to prevent legitimate users from using a service.
Remote to Local	Attacker does not have an account on the victim machine, hence tries to gain access.
User to Root	Attacker has local access to the victim machine and tries to gain super user privileges.
Probe	Attacker tries to gain information about the target host.

3.1.1 Problems with the DARPA Datasets

Capturing network traffic is a straightforward process and can be accomplished by placing a tool such as *tcpdump* on a host through which traffic on the network is routed. There is no shortage of publicly available network capture files which include numerous attacks (for example, the *pcapr*[1] project). However, such raw traffic is not particularly useful for researchers investigating IDSs as, without tagged attacks, analysis is not possible. To date, the DARPA datasets are the only publicly available, labeled datasets sufficiently large to perform research on. Unfortunately there are several significant problems with them.

The attack distribution in the DARPA datasets is very unbalanced. The vast majority of attacks are probes and denial of service instances. More interesting/serious attacks such as *phf* or *imap* are poorly represented. Because of this poor representation our tools were designed to extract proportionate samples of all attack classes, as we will explain in later sections. Comparing the DARPA traffic to real world network traffic highlights several serious flaws, as noted by [22, 31]:

- TCP SYN Regularity: In the DARPA traffic, there are always exactly 4 option bytes in inbound connections. In real traffic this value ranged from 0 to 28.
- Source Address Predictability: A small number of IP source addresses exist in the DARPA traffic, of which half are observed in 99% of the traffic. This distribution does not match real world traffic.
- Checksum Errors: A large proportion of packets have no checksum values.
- Packet Header Fields: The TTL and TOS values of the DARPA traffic are not representative of real world traffic. Only nine of the possible 256 values for TTL are observed, whilst 177 were unobserved in real traffic. For TOS 4 values were observed in the DARPA traffic, and 44 in real traffic.
- HTTP/SMTP: The DARPA HTTP requests and SMTP commands are all very regular, and not representative of real world requests.

Some authors warn that the training sets may even be *harmful* to research [10]. They note that the most fundamental flaw with the datasets is that no validation was ever performed to verify that the network traffic actually looked like real network traffic. The data rates in the datasets appear to be far too low to be representative of a real medium sized network and lack realistic Internet background noise. To highlight the contrived nature of the DARPA datasets, Mahoney and Chan noted that all attack traffic in the DARPA dataset had TTL values of 126 or 253, whilst normal traffic had TTL values of 127 or 254[23]. They suggested that this flaw was indicative of all experiments previously run on the datasets to be void.

Furthermore, the datasets contain outdated content and fail to include new threats. The DARPA datasets were innovative for their time but are now dated and not particularly representative of real world network traffic. However, they are still the standard benchmarking dataset for IDSs.

[1] http://pcapr.net

Table 3.2: TCP Traffic Summary

Category	Total TCP Attacks	Total TCP Attacks with Payload
DoS	1909762	1561
Warez	3347	2891
Probing	67009	1311
Exploit	5558	4497

3.1.2 Possible Alternatives

The majority of recent publications (similarly aware of the flaws of the DARPA datasets) include a separate data set of traffic generated or captured within networks they control to verify their results. Due to privacy reasons none of these institutions release their data to the public. Unfortunately, due to the short time scale of our project we were not able to generate our own datasets.

In July 2009 Sangster and co-workers released a paper documenting how network warfare competitions could be instrumental in generating large, scientifically valuable, modern labeled datasets [9]. They foresee their techniques as a possible alternative for the outdated DARPA datasets. Dozens of network warfare competitions are held every year and may contain elements for generating useful datasets. The authors do not claim that their techniques can provide more value than the DARPA dataset. However, they do argue that over time it may be possible to move towards wargames that provide more realistic network traffic and therefore provide researchers with datasets of higher quality than those currently available. As part of their research they released the network traffic and labels from the 2009 4-day Cyber Defense Exercise competition held between the National Security Agency (NSA) and the U.S. Military Academy. Their techniques and datasets may indeed be the future benchmark for IDSs, but due to their comparatively late release we were unable to include them in our research.

3.1.3 Other Constraints

Since the core part of our research focused on analyzing the payload of TCP content streams, we required training and testing samples of TCP traffic with payload content. Selecting only TCP traffic from the DARPA dataset reduced the number of samples we had. Furthermore, selecting only TCP traffic with payload content reduced the number of samples by a further significant proportion. We discovered that the majority of TCP attacks contained no payload, but that being said, a lack of payload may well be indicative of an attack (many DoS attacks consist only of the connection setup packets). We chose to evaluate on TCP traffic, which was predominantly payload based, but also included an amount of TCP traffic without payload. The summary of the samples available after preprocessing and insertion into our database are presented in Table 3.2.

3.2 Data Preprocessing

We developed a set of tools to extract various features from the DARPA network dump files. All programing was done in the Perl programming language and a backend database was deployed (MySQL) for easy manipulation of and access to the features. Much of the heavy lifting was done

by making use of existing tools within our scripts, namely; netdude [19], tcptrace [24] and tcpflow [33]. Several more scripts were designed for easy selection of random sets of data from the database and for their conversion into the Weka .arff file format. Figure 3.2 provides a high level overview of the process, which occupied a significant amount of time for the project.

3.2.1 Definition of a Flow

Within TCP/IP, a flow may be defined as "one or more packets traveling between two computer addresses and a particular pair of ports (defined for each end of the flow)" [6]. This tuple of $(host_{src}, host_{dst}, port_{src}, port_{dst})$ can identify every TCP packet belonging to a particular flow. The source and destination parts of the tuple would be reversed for traffic flowing in the opposite direction. Since TCP is a stateful protocol [29], its flows have clearly defined start and end points (i.e. connection establishment and breakdown).

3.2.2 Demultiplexing

The DARPA datasets provide large network dump files ranging in size from 100MB to 2GB. Each file contains all traffic between all hosts for the period of one day. For our 6 weeks of data we had a total of 19.5GB worth of raw network traffic. Our first objective was to extract individual TCP sessions from these large network dump files. This process is generally referred to as *demultiplexing*. A TCP session is established via a 3-way handshake which consists of the client sending a packet with the SYN flag set, the server responding with the SYN and ACK flags set and the client responding with the ACK flag set. From this point on all packets matching the tuple $(ip_{src},, ip_{dst}, port_{src}, port_{dst})$ belong to this particular session. Sequence numbers ensure packets are reassembled in the correct order and a similar process to the 3-way handshake is used for tearing down the connection (using FIN instead of SYN flags).

In order to reconstruct TCP sessions from raw packet data it is necessary to traverse the file in a linear fashion, examining each IP packet individually. We developed a tool called *flow_parser.pl* which handles most of the data manipulation and feature extraction process. It is a command line perl script with numerous options, as are presented in Table 3.3. The *-dump* flag demultiplexes a dump file creating individual dump files for each session using the *netdude* tool, whilst the *-dir* option specifies the directory to search for the raw dump files. For example, the command:

```
$perl flow_parser.pl -dir ./DARPA/1998/ -dump
```

creates a folder with the name *<dumpfilename>_demux* for each dump file found. Under this directory subfolders are created to denote the protocol number found in the IP header of the communication. For example, a subfolder with the name *p6* indicates TCP sessions are contained within it as the number 6 is the RFC791 [28] defined value for TCP. We also observed instances of ICMP (1) and UDP (17) in the training data, but these were ignored. Under each *pn* subfolder a further hierarchy is created to denote source and destination IP addresses as well as source and destination ports. Finally inside each of these subfolders exist individual *.trace* files containing only the packets for a single session. For example:

```
./week4/tuesday/demux_outside/p6/S197.182.91.233/D172.16.114.50/
```

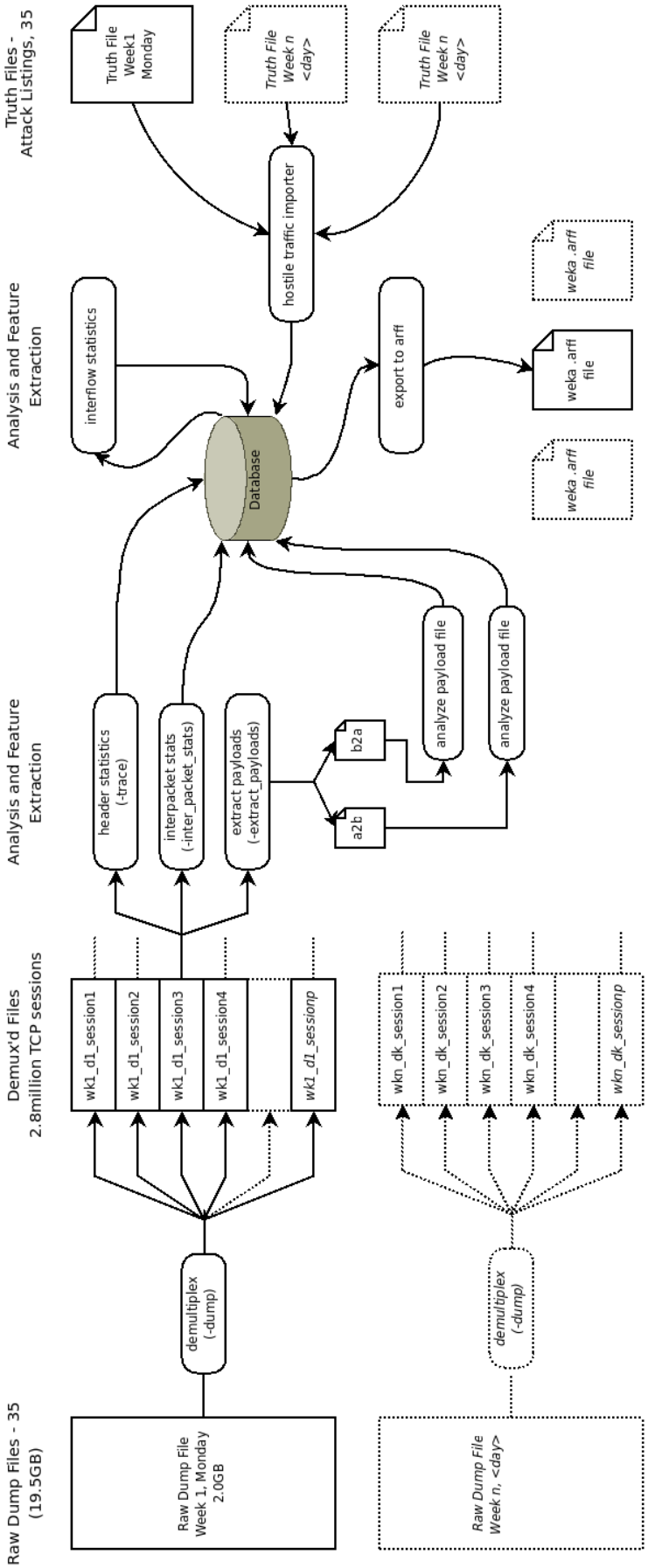


Figure 3.2: Overview of Feature Extraction Process

Table 3.3: flow_parser.pl script options

Flag	Description
-dir	Specifies the directory in which to search for dump files, trace files or payload files (depending on what other flags are given).
-dump	Demultiplex all raw dump files into individual TCP session trace files. Individual files are placed in the directory ./demux_<*filename*>/p6/S<s*rc-ip*>/D<*dst-ip*>/ the trace filenames have the structure: <*epochtime*>-s<*src-port*>-d<*dst-port*>.trace
-trace	Search for all *.trace* files in the specified *-dir* directory and apply the *tcpt*race program to them. Capture the results and insert them into the specified table in the SQL database.
-rtt	Additional Round Trip Time statistics which can be included with the *-trace* option.
-tbl_as_filename	Create a new table for every raw dump file with the name of the filename.
-tbl	Insert all data into the same table, as per the name after the flag.
-darpa_to_sql	Checks the location of the file and calculates the correct table from the file and directory structure. e.g ./data/**week2**/**wednesday**/demux_wednesday /.../foo.trace will have its data stored in the table *week2_wednesday.*
-tbl_prefix	Prefix for the table name for one of the above options.
-tbl_postfix	Postfix for the table name for one of the above options.
-inter_packet_stats	Calculate interpacket statistics on all *.trace* files found under the *-dir* directory and insert their values into the specified table.
-extract_payloads	Extract the TCP payloads from all *.trace* files found under the *-dir* directory.
-analyze_payloads	Analyze all extracted payload files under the *-dir* directory and insert the calculated values into the specified table.

denotes a folder containing all TCP sessions in individual *.trace* files which in turn contain all packets from host *197.182.91.233* to host *172.16.114.50*. Each separate .trace file has a filename of the form <first-packet-arrival>-<src-port>-<dst-port>.trace. For example, the file:

```
898641295.793307-s3140-d80.trace
```

within the above mentioned folder denotes a session from host *197.182.91.233* with source port 3140 to host *172.16.114.50* with destination port 80. This example would most likely be web traffic, since the destination port is 80. Our 6 weeks of traffic (19.5 GB) was demultiplexed into 2,859,801 individual files[2], each containing packets representing a single TCP session. These 2.8 million trace files were comprised of 28,234,762 packets in total. It took 7 hours and 37 minutes for our script to demultiplex the 19.5GB worth of network traffic.

3.2.3 Payload Extraction

The *flow_parser.pl* script is further used to extract the TCP payload from the network flows of each individual session file (created in the previous step). We modified the *tcpflow* program and incorporated it into our script to perform this task. Datastreams are reconstructed from individual packets and stored in separate files for later analysis. Sequence numbers are correctly understood and the data streams are extracted regardless of out of order delivery or retransmission. Running the following command extracts payloads for all *.trace* files located under the *./DARPA/1998/* directory and all subdirectories:

```
$perl flow_parser.pl -dir ./DARPA/1998/ -extract_payloads
```

A pair of files is created for each session- one for each direction of the flow of data between the two hosts involved in the communication (i.e. client to server, and server to client). Files containing the data stream contents are output with filenames of the form:

<timestamp>–<ip–src>.<src–port>–<ip–dst>.<dst–port>

Looking at our previous example, within the directory:

```
./week4/tuesday/demux_outside/p6/S197.182.91.233/D172.16.114.50/
```

We note the raw file *898641295.793307-s3140-d80.trace* has had its stream data extracted into two files, one for each direction of the traffic creating the files:

1. `898641295.793307-172.016.114.050.00080-197.182.091.233.03140`

2. `898641295.793307-197.182.091.233.03140-172.016.114.050.00080`

These files contain stream data from host *197.182.91.233* with source port 3140 to host *172.16.114.50* with destination port 80 and vice versa. Inspecting the contents of these extracted payload files, we see that it was indeed a web request/response as can be seen in Figures 3.3 and 3.4.

[2]Not including non TCP traffic files

```
GET /images/home.gif HTTP/1.0
Referer: http://marx.eyrie.af.mil/faq/callsign.html
User-Agent: Mozilla/3.01 (X11; I; SunOS 4.1.4 sun4u)
Host: marx.eyrie.af.mil
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

Figure 3.3: HTTP GET Request, 197.182.91.233:3140 - 172.16.114.50:80

```
HTTP/1.0 200 OK
Date: Tue, 23 Jun 1998 22:35:03 GMT
Server: Apache/1.1.3
Content-type: image/gif
Content-length: 2068
Last-modified: Tue, 03 Jun 1997 16:44:52 GMT

GIF87a22÷[      T<8d>b<8c>¾§¾¾<8b>»^^UTuGq<97>L<93><97>p<96>éÚæp^Xl¯<95>¬|R{éÃá¯k¾^?
¸<84>»}f&`Ú¦Ô<97>X<95>úéòp$mf      [f^Aaé¹ç<84>H<82>Ú¬Øf^Sbu@rÙ<99>Úf^Ycf _¢|<9e>¯m¥¾^?
¸<97>^<94>p*lulnùáñ<84>Y<83>éÁè¯<8c>®<97>M<8d>¯yù×øù÷ñj
dù.÷b^CZp^Zf<97>A<90>j*hj        _p^Ajp^Nej^T`<8d>P<8b>Ë{ËË<89>È¢
g<9f>ÚµÖË§Ç<8d>V<8a>ùÏø<84>C¬ù²ój^Acé²ç¾w»¯e|lr¢o<9a>ùßùË<9e>Ëj `é²ç¢^¢ùóùu
$qùÀ÷<8d>K<86>j^SfÚ<9f>Õj^Z_¢n ¢a u)rôëùùÈ÷Ë<97>É¢V<9a>j^[i^^HZp7jË<90>É|Nx¢
N<9f>u^Zmb^M^éÊå¾pºË<82>É<8d>D<89><97>Y<91>ùñöf^O]f^C_|Azé¢å<84>O<80>¾<86>.|Oyùéö<8d>
\<8b><97>S<90>ùýúb^G]p ij^Phj^Zc<8d>Q<87>|Gy<97>j<96>¾<94>.p/
n¾<9b>¹<84>T<82>¯<81>²<8d>E<86><84>B<82>f^\ep^ThË<8b>ÆÚ»Ù<8d>W<89>é´ã¢v<9d>ùáõj%cj^Taj d¢
d¢¾<8d>·p^Zgp%jé»ä<84>I~ÚÕÚ<9d>Ø<97>`<92>p+iéÃåùÙõùºôj+dùÑôj^Ca¾x¶Ë Æu%nùÅó¢o<9e>u
+oùÊöË<99>Å^      W¢O<9b>b^O[¾quË<84>Ä|lv<8d>]<89>øïùf^Mb¯r¬¾<84>»øçùù÷ùb^D`p^^lj^Mdj^Xf|
5x¢t j$fj^^g<97>[<97>f^Dd<83>Q<84>®|¯Ê<8d>Ë<8c>Y<8d>æµéõãùèëéj^Ui,22^Hþ<83>^H^\H
° Å<83>^H^S^Vж^Q<84>¡Å<86>^P^_><8c>^H<91>¢D<8b>^X^U&d¨±ãB
```

Figure 3.4: Sample GET Response, 172.16.114.50:80 - 197.182.91.233:3140

3.3 Feature Extraction

Having demultiplexed all TCP sessions into individual trace files and having extracted the payloads from each of these files, we moved on to generating statistical features from them. Our aim was to initially extract as many features as possible, and then narrow them down before the classification stage. Before discussing this process, we present a short discussion on TCP/IP headers. Whilst the exact meaning of the options in the headers may be irrelevant, since our classification process will simply look for patters concerning them, we nonetheless provide a brief explanation of the protocols for the reader unfamiliar with them.

3.3.1 TCP/IP Headers

The Open Systems Interconnection (OSI) model divides network architecture into seven layers, with each layer broadly containing a different abstraction. When sending data, each layer encapsulates the previous layer until reaching the bottom layer. The receiver reverses this process, decapsulating each layer until reaching the top layer. The coupling of TCP/IP is the standard method for reliable communication on the Internet.

RFC791 [28] specifies the Internet Protocol (IP) as a *packet protocol*. IP is found at the Network Layer (layer 3) of the OSI stack and has the task of delivering packets from a source host to a destination host. It is a connectionless protocol which means no state is maintained. The IP header is depicted in Figure 3.5. Briefly examining the header fields; **Version** denotes the IP version number- all of our data is IPv4. Internet Header Length (**IHL**) represents the length of the header, and therefore points to the start of the data. **Type of Service** is intended to give an indication of the Quality of Service (QoS) desired, but is seldom used or implemented. **Total length of packet** represents the length of the header plus the data section of the packet. We note that it allows a packet to have a size up to 65,535 octets, but such packets are impractical. **Identification** is a 16 bit value that is common to each fragment belonging to a particular message and aids in assembling

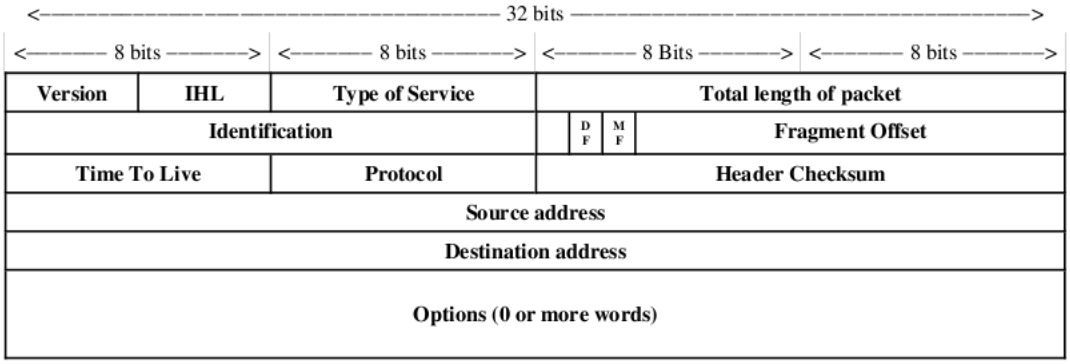


Figure 3.5: IP Headers [32]

fragments of a datagram. The control flags **DF** (Don't Fragment) and **MF** (More Fragments) and the **Fragment Offset** are used to reassemble fragmented datagrams. **Time To Live** (TTL) specifies how many nodes the packet can pass through before being dropped- its value is decremented by each node it passes through. **Protocol** specifies the higher level protocol as per RFC 1700 [30]. For example, TCP is number 6, as was mentioned in Section 3.2.3. **Header Checksum** is calculated against the header to protect against corruption during sending but is somewhat superfluous due to lower level checksums. **Source address** and **destination address** specify 32 bit addresses for the originator and recipient of the packet- these values are used by routing infrastructure to route packets from source to destination (packets of the same communication may take different paths). **Options** specifies various additional options and space for future development, but is seldom used.

TCP is encapsulated within IP and is found at the Transport layer (4) of the OSI model. It is defined in RFC 793 [29] and is responsible for the guaranteed delivery of packets. It controls the rate at which data is sent, the size of segments, ordered reassembly and is responsible for congestion management. We examine the header fields of TCP as depicted in Figure 3.6. The concept of ports is used in TCP, with one port on each end of a connection. Packets arriving at a host have their session identified by the port numbers. The **source port** and **destination port** are 16 bit numbers. The **sequence number** and **acknowledgment number** are used to ensure packets are reassembled in the correct order. The **header length** specifies the length of the header, and therefore points to the start of the data. Several flags are available; **URG** represents the urgent flag. It is up to the receiving host to interpret this flag- it may be used for QoS or, more commonly, simply ignored. The **ACK** flag indicates that the acknowledgment number field is significant and in combination with the **SYN** and **FIN** flags is used for connection establishment and tear down. **RST** indicates the session should be reset. The **PSH** (push) flag indicates the receiving host should not buffer the received data but immediately push it to the application. The **URG** flag and the **Urgent Pointer** are used to override the usual first-in-first-out sequence of sending data, and have the packet sent to the front of the queue (which is useful for error messages or canceling operations).

3.3.2 Header Related Features

Our *flow_parser.pl* script makes use of the *tcptrace* program to generate statistics from the TCP and IP headers (mentioned in Section 3.3.1) of all packets in each flow and insert the results into our database. This is induced with the *-trace* and *-darpa_to_sql* options as the following command

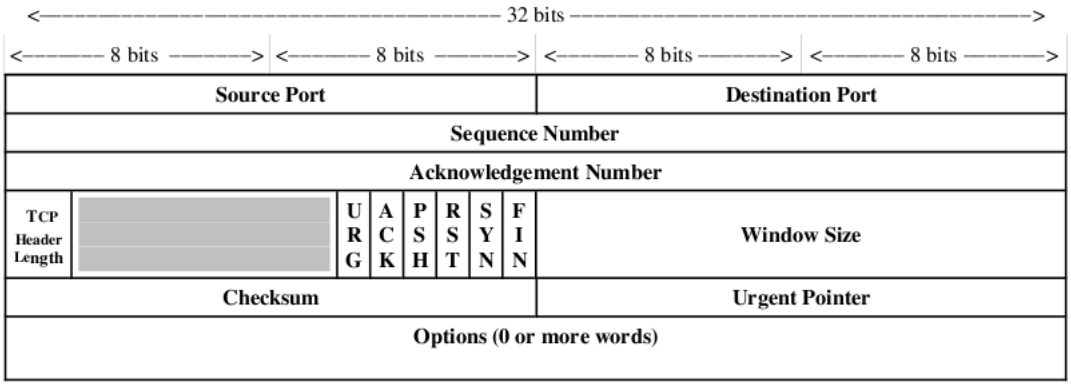


Figure 3.6: TCP Headers [32]

illustrates:

```
$perl flow_parser.pl -dir ./DARPA/1998/ -darpa_to_sql -trace
```

The *-trace* option extracts statistical features from the headers, whilst the *-darpa_to_sql* option indicates that the output should be inserted into a table matching the DARPA hierarchy created by the *dump* option. For example, when the script examines the file *898641295.793307-s3140-d80.trace* in the directory:

```
./week4/tuesday/demux_outside/p6/S197.182.91.233/D172.16.114.50/
```

it will generate header statistics for the *.trace* file and the output will be inserted into the newly created table **week4_tuesday**.

A total of 92 features were processed and extracted from the header fields for each TCP session- of these, 85 relate to the bidirectional nature of the communications. For example, *total_packets_a2b* and *total_packets_b2a*. Additionally, the source address (host_a), destination address (host_b), source port (port_a), destination port (port_b) and first/last packet arrival times of the stream are extracted. A description of all the bidirectional features is presented in the Table 3.5. The descriptions in this table are from the *tcptrace* manual[24].

Once all the features have been imported into the database the script creates an index against *(first_packet, host_a, host_b, port_a, port_b)* to facilitate fast searching and updating. The index dramatically decreased the amount of time operations on the table took. The cardinality of the elements for a table containing all 35 days of traffic is presented in Table 3.4. The total time to extract the statistics from our 2,859,801 individual session files and insert them into the database was 20 hours and 40 minutes.

Table 3.4: Index Cardinality

Column Name	Cardinality
first_packet	285992
host_a	2859924
host_b	2859924
port_a	2859924
port_b	2859924

Table 3.5: Header Features

Name (a2b + b2a)	**Description**
total packets	The total number of packets exchanged between the two hosts.
ack pkts sent	The total number of ack packets seen (packets with the TCP *ACK* bit set).
pure acks sent	The total number of ack packets sent between the hosts that were not piggy-backed with data (just the TCP header and no TCP data payload) and did not have any of the SYN/FIN/RST flags set.
sack pkts sent	The total number of ack packets sent carrying TCP SACK [??] blocks.
dsack pkts sent	The total number of sack packets seen that carried duplicate SACK (D-SACK) [6] blocks.
max sack blks/ack	The maximum number of sack blocks seen in any sack packet.
unique bytes sent	The number of unique bytes sent, i.e., the total bytes of data sent excluding retransmitted bytes and any bytes sent doing window probing.
actual data pkts	The count of all the packets with at least a byte of TCP data payload.
actual data bytes	The total bytes of data seen. Note that this includes bytes from retransmissions / window probe packets if any.
rexmt data pkts	The count of all the packets found to be retransmissions.
rexmt data bytes	The total bytes of data found in the retransmitted packets.
zwnd probe pkts	The count of all the window probe packets seen. (Window probe packets are typically sent by a sender when the receiver last advertised a zero receive window, to see if the window has opened up now).
zwnd probe bytes	The total bytes of data sent in the window probe packets.
outoforder pkts	The count of all the packets that were seen to arrive out of order.
pushed data pkts	The count of all the packets seen with the PUSH bit set in the TCP header.
SYN/FIN pkts sent	The count of all the packets seen with the SYN/FIN bits set in the TCP header respectively.

Continued on next page

req 1323 ws/ts	If the endpoint requested Window Scaling/Time Stamp options as specified in RFC 1323[9] a 'Y' is printed on the respective field. If the option was not requested, an 'N' is printed. For example, an "N/Y" in this field means that the window-scaling option was not specified, while the Time-stamp option was specified in the SYN segment.
adv wind scale	The window scaling factor used. Again, this field is valid only if the connection was captured fully to include the SYN packets. Since the connection would use window scaling if and only if both sides requested window scaling [9], this field is reset to 0 (even if a window scale was requested in the SYN packet for this direction), if the SYN packet in the reverse direction did not carry the window scale option.
req sack	If the end-point sent a SACK permitted option in the SYN packet opening the connection, a 'Y' is printed; otherwise 'N' is printed.
sacks sent	The total number of ACK packets seen carrying SACK information.
urgent data pkts	The total number of packets with the URG bit turned on in the TCP header.
urgent data bytes	The total bytes of urgent data sent. This field is calculated by summing the urgent pointer offset values found in packets having the URG bit set in the TCP header.
mss requested	The Maximum Segment Size (MSS) requested as a TCP option in the SYN packet opening the connection.
max segm size	The maximum segment size observed during the lifetime of the connection.
min segm size	The minimum segment size observed during the lifetime of the connection.
avg segm size	The average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data pkts reported.
max win adv	The maximum window advertisement seen. If the connection is using window scaling (both sides negotiated window scaling during the opening of the connection), this is the maximum window-scaled advertisement seen in the connection. For a connection using window scaling, both the SYN segments opening the connection have to be captured in the dumpfile for this and the following window statistics to be accurate.
min win adv	The minimum window advertisement seen. This is the minimum window-scaled advertisement seen if both sides negotiated window scaling.
zero win adv	The number of times a zero receive window was advertised.

Continued on next page

avg win adv	The average window advertisement seen, calculated as the sum of all window advertisements divided by the total number of packets seen. If the connection endpoints negotiated window scaling, this average is calculated as the sum of all window-scaled advertisements divided by the number of window-scaled packets seen. Note that in the window-scaled case, the window advertisements in the SYN packets are excluded since the SYN packets themselves cannot have their window advertisements scaled, as per RFC 1323
initial window bytes	The total number of bytes sent in the initial window i.e., the number of bytes seen in the initial flight of data before receiving the first ack packet from the other endpoint. Note that the ack packet from the other endpoint is the first ack acknowledging some data (the ACKs part of the 3-way handshake do not count), and any retransmitted packets in this stage are excluded.
initial window packets	The total number of segments (packets) sent in the initial window as explained above.
ttl stream length	The Theoretical Stream Length. This is calculated as the difference between the sequence numbers of the SYN and FIN packets, giving the length of the data stream seen. Note that this calculation is aware of sequence space wrap-arounds, and is printed only if the connection was complete (both the SYN and FIN packets were seen).
missed data	The missed data, calculated as the difference between the ttl stream length and unique bytes sent.
truncated data	The truncated data, calculated as the total bytes of data truncated during packet capture.
truncated packets	The total number of packets truncated
data xmit time	Total data transmit time, calculated as the difference between the times of capture of the first and last packets carrying non-zero TCP data payload.
idletime max	Maximum idle time, calculated as the maximum time between consecutive packets seen in the direction.
throughput	The average throughput calculated as the unique bytes sent divided by the elapsed time i.e., the value reported in the unique bytes sent field divided by the elapsed time (the time difference between the capture of the first and last packets in the direction).

3.3.3 TCP Datastream Payload Features

Stefano Zanero in his research into unsupervised learning based on individual packet payloads noted that an algorithm should be able to "sensibly" classify payloads. He notes that techniques which analyze content should (i) preserve as much information as possible about the similarity

of payloads, (ii) separate traffic from different protocols into different groups and (iii) separate attack traffic from normal traffic [31]. Although he was referring to packet payloads, his advice is nonetheless applicable to our research.

The process for extracting stream content from each TCP session was described in Section 3.2.3. We now explain the features calculated from the session payloads. The *-analyze_payloads* flag in the *flow_parser.pl* script performs this task:

```
$perl flow_parser.pl -dir ./DARPA/1998/ -darpa_to_sql -analyze_payloads
```

The payload analysis includes n-gram and entropy calculations which we now discuss.

n-grams

An n-gram is a sequence of n tokens. Their use if fairly common in areas such as natural language processing, information retrieval and the sequence analysis of genomes. An n-gram of size one is known as a unigram, of size two a bigram, of size three a trigram and so on. n-grams are generated by sliding a window of length n over a list of items [16]. As a simple example, consider the sequence of words "all work and no play". Considering individual words to be tokens, all possible bigrams generated from this sequence are $\{$*all work, work and, and no, no play*$\}$.

In our context, tokens are single byte values represented by an integer (i.e. byte c, $0 \leq c \leq 255$). Therefore a unigram model simply consists of a count of each character observed and can be represented by a histogram of size 256. Expanding this to a bigram model we obtain a list of every observed two sequence combination. For the 256 possible byte values, this results in an alphabet of size 256^2.

For each payload file, we calculate the unigram and bigram frequencies. From this count, we can calculate the unigram and bigram entropy.

Entropy

The formal name for the average information per symbol in a set of symbols is called entropy [15]. The byte characters in our payload files have a frequency distribution and the information of any byte symbol can be represented by $log_2(n)$ [15]. If all byte characters in the files were equally likely, the probability of any particular character would simply be $\frac{1}{256}$. However, since the distribution is not flat, we weigh the information of each byte character by its probability of occurring. This is known as Shannon's Entropy:

$$H = -\sum_{i}^{n} p_i log_2 p_i$$

We can calculate the unigram entropy for a payload file as presented in Algorithm 1. We can expand the unigram entropy calculation to that of bigram entropy calculation by counting every sequence of two characters. This is presented in Algorithm 2.

Apart from calculating the unigram and bigram entropy, we also keep track of the most frequent bi/trigram and their count, the number of zeroes, number of printable/nonprintable characters and

Algorithm 1 Unigram Entropy Calculator

```
my %Count;
my $file = read_file($filename, binmode => ':raw' );
my $total=length($file);
my $H=0;

foreach my $char (split(//,$file))
{
   $Count{$char}++;
}

foreach my $char (keys %Count)
{
   my $p = $Count{$char}/$total;
   $H += $p * log($p);
}
$H = -$H/log(2);
```

Algorithm 2 Unigram and Bigram Entropy Calculation

```
my %bigram_count=();
my %unigram_count=();
my $H_bi=0;
my $H_uni=0;
my $file = read_file($filename, binmode => ':raw' );
$total_chars=length $file;

for(my $i=0; $i<$total_chars-1; $i++) {
    my $bigram = substr($file, $i, 2);
    my $unigram = substr($file, $i,1);
    $bigram_count{$bigram}++;
    $unigram_count{$unigram}++;
}
$unigram_count{substr($file, $total_chars-1, 1)}++;

foreach my $uni(keys %unigram_count) {
    my $p=$unigram_count{$uni}/$total_chars;
    $H_uni+=$p*log($p);
}
$H_uni=-$H_uni/log(2);

foreach my $bi(keys %bigram_count) {
        my $p=$bigram_count{$bi}/$total_chars;
        $H_bi+=$p*log($p);
}
$H_bi=-$H_bi/log(2);
```

Table 3.6: Payload Features (a2b)

Name	Description
total_chars_a2b	Total number of characters observed in the flow.
unigram_entropy_a2b	Shannon's entropy calculated over the alphabet of all ASCII characters (0-255).
bigram_entropy_a2b	Shannon's entropy calculated over the alphabet of all bigram combinations of ASCII characters (i.e. $0\text{-}255^2$).
most_freq_char_a2b	Most frequent unigram observed in the flow. Represented as its ASCII ordinal value.
count_mfc_a2b	Number of times the most frequent unigram was observed.
prop_mfc_a2b	Proportion of the most frequent unigram to the total number of characters observed.
mf_bigram_a2b	Most frequent bigram observed in the flow. Represented as a pair of ASCII zero padded ordinal values, e.g {071013} for {G,\n}.
count_mf_bigram_a2b	Number of times the most frequent bigram was observed.
prop_mf_bigram_a2b	Proportion of the most frequent bigram to the total number observed.
num_of_zeroes_a2b	Number of zeroes observed in the flow.
prop_zeroes_a2b	Proportion of zeroes to the total number of characters observed in the flow.
num_printable_a2b	Number of printable characters observed. i.e. ASCII values in the range $32 \leq value \leq 126$
num_non_printable_a2b	Number of non-printable characters observed. i.e. ASCII values in the range $0 \leq value \leq 31 \; and \; 127 \leq value \leq 255$
prop_printable_a2b	Proportion of printable characters to the total number of characters observed in the flow.
prop_non_printable_a2b	Proportion of non-printable characters to the total number of characters observed in the flow.

Table 3.7: Interpacket Features

Name	Description
inter_arrival_time_mean_a2b	Mean of packet inter arrival times.
inter_arrival_time_variance_a2b	Variance of packet inter arrival times.
inter_arrival_time_stdev_a2b	Standard deviation of packet interarrival times.
payload_size_mean_a2b	Mean of packet payload sizes.
payload_size_variance_a2b	Variance of packet payload sizes.
payload_size_stdev_a2b	Standard deviation of packet payload sizes.

all their proportions. These features are listed with explanations in Table 3.6. This table only lists features for the direction of client to server (a2b)- all features are repeated for server to client (b2a).

It took 2 hours and 50 minutes for our script to analyze all the extracted payload files and insert the metrics into our database.

3.3.4 Interpacket Features

The next set of features we examine relate to characteristics of packets within a flow. The *flow_parser.pl* script performs this analysis with the option *-inter_packet_stats* as follows:

```
$perl flow_parser.pl -dir ./DARPA/1998/ -darpa_to_sql -inter_packet_stats
```

This calculates interpacket relations regarding arrival times and payload sizes. The packet inter-arrival times are defined as the difference in arrival time between the n and $n+1$ packets within a flow. These features are quite similar to those created by the *-trace* option (Section 3.3.2) as they are calculated from header values. The interpacket statistics and their descriptions are presented in 3.7.

It took 18 hours and 21 minutes for our script to analyze all 2,859,801 raw session files and insert the metrics into our database.

3.3.5 Interflow Features

Interflow features are calculated by examining flows in the database with the script *interflow_analysis.pl*. This script can be executed as follows:

```
$perl perl interflow_analysis.pl -prefix "week\d_(monday|tuesday|wednesday
    |thursday|friday)"
```

The *-prefix* options uses regex to match the tables we wish to perform the analysis on. The above example matches all our data. A flow can be uniquely defined as the tuple $(ip_{src}, ip_{dst}, port_{src}, port_{dst})$. We examine each row (i.e. flow) in the database and count the number of several characteristics observed in the past 10 seconds. These are:

1. Total Connections between the two hosts in the last 10 seconds on any port

Table 3.8: Interflow Features

Name	Description
flow_duration	The length of time the session lasted. It is calculated as the time stamp of the last packet minus the time stamp of the first packet.
total_conns_10s_anyport	Performs a count against all flows in the last 10 seconds matching the same (ip_{src}, ip_{dst})
total_conns_10s_sameServerPort	Performs a count against all flows in the last 10 seconds matching the same $(ip_{src}, ip_{dst}, port_{dst})$
total_packets_a2b_10s	Sums all packets between (ip_{src}, ip_{dst}) in the last 10 seconds (from a 2 b).
total_packets_b2a_10s	Sums all packets between (ip_{src}, ip_{dst}) in the last 10 seconds (from b 2 a).

2. Total connections between the two hosts in the last 10 seconds on the same server port

3. Total packets sent between the two hosts in the last 10 seconds (in both directions, a2b and b2a)

Additionally, we calculate the flow duration for each flow. These features are summarized in Table 3.8. It took 15 hours and 38 minutes for the script to calculate all interflow statistics.

3.3.6 Attack Features

Along with the raw tcpdump files (as discussed in Section 3.2.2) the supplied data contains *truth tables*, which list attack names matched to traffic instances in plain text files. This data allows researchers to identify hostile traffic from regular traffic in the raw tcpdump files supplied. The tables contain the following space separated columns: identity number, date, time of first packet, duration, protocol, source port, destination port, source address, destination address, attack score and attack name for every instance of network traffic, both regular and attack. A sample of this data is presented in Table 3.10. In the first row, we observe an *rlogin* attack at *17:00:05* against machine *192.168.0.20* from an attacker using the machine with an address of *192.168.1.30.* An attack score of 0 and an attack name of '-' as observed in rows 2 and 9 indicates regular traffic. We observe the attacker using machine 192.168.1.30 performing a *port-scan* against machine *192.168.0.20* as he sequentially checks a series of well known ports (from rows 4 to 8). The *Protocol* field denotes TCP traffic unless it is tagged with one of the modifiers /i or /u, for ICMP traffic and UDP traffic respectively. We see an example of an attack over ICMP in row 12- the *satan* attack. As our research was only concerned with TCP traffic, such rows were ignored during analysis.

We developed a script called *hostileTrafficImporter.pl* to parse all 35 of these flat files, import them into our database and match them to our previously imported raw traffic features. Each one is inserted as a separate table, for example *week4_tuesday_attackList.* The script then matches $(ip_{src}, ip_{dst}, port_{src}, port_{dst})$ of the attack list to the $(ip_{src}, ip_{dst}, port_{src}, port_{dst})$ in the feature table.

This script is executed as follows overleaf.

Table 3.9: Attack Features

Name	Description
attack_score	0 indicating no attack, 1 indicating attack. Used for binary classification.
attack_name	One of the 67 observed attack names. or "-" for normal traffic.
attack_class	One of the classes probing, exploit attempt, warez transfer, Denial of Service or normal. Used for multiclass classification.

```
$perl hostileTrafficImporter_new.pl -dir ./DARPA/1998/ -darpa -join_tables
    -join_prefix "week\d_(monday|tuesday|wednesday|thursday|friday)"
```

The *-prefix* options specifies a search pattern to match- for every *week<n>_<day>* found, it expects a corresponding *week<n>_<day>_attackList* table. Every row in the feature database is updated with values for attack_score and attack_name. Table 3.11 lists information about the tagging of the raw data. As can be seen, over 99% of the raw traffic was successfully tagged.

As previously mentioned, we classified all attacks into one of five classes, namely: probing, exploit attempt, warez transfer, DoS or normal traffic. This is represented by the attack_class column. These features are summarized in Table 3.9.

Parsing and importing the 35 truth files into our database took 25 minutes.

Table 3.10: Labeled Network Traffic

Num	Date	Time	Duration	Protocol	Source Port	Destination Port	Source IP	Destination IP	Attack Score	Attack Name
128	01/23/1998	17:00:05	00:00:14	rlogin	1022	513	192.168.1.30	192.168.0.20	1	rlogin
...	...	...	...	...	...	...	...	...	...	...
258	01/23/1998	17:03:00	00:00:10	http	2019	80	192.168.1.30	192.168.1.40	0	-
259	01/23/1998	17:03:52	00:00:05	telnet	2020	23	192.168.1.30	192.168.0.20	1	port-scan
260	01/23/1998	17:03:53	00:00:02	ssh	2021	22	192.168.1.30	192.168.0.20	1	port-scan
261	01/23/1998	17:03:53	00:00:02	ftp	2022	21	192.168.1.30	192.168.0.20	1	port-scan
262	01/23/1998	17:03:54	00:00:01	finger	2023	79	192.168.1.30	192.168.0.20	1	port-scan
263	01/23/1998	17:03:54	00:00:05	http	2024	80	192.168.1.30	192.168.0.20	1	port-scan
...	...	...	...	...	...	...	...	...	...	...
310	01/23/1998	17:04:53	00:00:01	smtp	1048	25	192.168.1.30	192.168.1.20	0	-
...	...	...	...	...	...	...	...	...	...	...
619	01/23/1998	18:06:33	00:00:01	eco/i	-	-	207.230.054.203	172.016.114.050	1	satan

Table 3.11: Tagging of Attacks

		Attack List Table (from flat files)		Extracted Session Tables (from raw traffic)		
Week	Day	Listed TCP Attacks	Listed Other Attacks (Ignored)	Tagged Sessions	Total Sessions	Successfully Tagged
Dataset One						
1	Mon	13494	118504	12921	14410	89.67%
	Tue	18188	6943	17867	18918	94.44%
	Wed	9306	3905	9303	9448	98.47%
	Thu	14069	33501	12876	15191	84.76%
	Fri	6556	9724	6482	6721	96.44%
2	Mon	10728	22264	10669	11226	95.04%
	Tue	11188	12947	11085	11431	96.97%
	Wed	10614	34084	10448	10952	95.40%
	Thu	210600	16566	211151	211283	99.94%
	Fri	252800	34192	253416	253794	99.85%
3	Mon	11654	37306	11330	12048	94.04%
	Tue	15125	35761	13861	14821	93.52%
	Wed	230761	36320	231307	231404	99.96%
	Thu	510630	33992	511832	511945	99.98%
	Fri	39077	44116	38951	39040	99.77%
4	Mon	43095	1269	43028	43078	99.88%
	Tue	33143	1499	32854	32957	99.69%
	Wed	47501	2071	47197	47437	99.49%
	Thu	47296	10089	47178	47454	99.42%
	Fri	243820	6508	244231	244315	99.97%
Dataset Two						
1	Mon	40076	19403	40152	40214	99.85%
	Tue	258909	12981	259573	259963	99.85%
	Wed	45194	16351	44779	44831	99.88%
	Thu	42611	76707	42205	42272	99.84%
	Fri	244380	60317	207769	208032	99.87%
2	Mon	56477	77304	55081	56146	98.10%
	Tue	76451	4583	64632	65150	99.20%
	Wed	50043	56871	49541	49650	99.78%
	Thu	260246	32747	259566	260413	99.67%
	Fri	45628	63203	45296	45380	99.81%

3.3.7 Justification for Feature Choices

It is fairly common for an attack to be based on malformed **headers** in either the TCP or IP fields of one or more packets[3]. The protocol stack of a particular operating system may not be programmed to handle such a header, which can result in either a system crash or the attacker being able to inject code for arbitrary execution. For example an exploit[4] was discovered in 2005 within the Microsoft operating systems such that when processing an IP packet with an options size of 39 the operating system would crash. The maximum value for the IP options field is 40 and two are already used. As we are studying TCP flows as opposed to individual packets we chose to create features composed of all the packets.

Some attacks leave signatures in the **interpacket** characteristics of flow. For example, very regular intervals of packet arrival times in a stream may be indicative of an attack.

[3]In fact, such attacks are found in fields other than networking too. Malformed headers in movie, music and document files have been used to either crash computers or take control of them via execution of arbitrary code.

[4]www.milw0rm.com/exploits/942

Some attacks can be identified by looking at several TCP sessions. For example, a DoS attack or a probing attack would most likely comprise of many connections between two hosts over a period of time. This type of attack may be detected by examining how many times two hosts have communicated. We therefore chose to compute several characteristics regarding **interflow** statistics. A recent example of such an attack is a tool developed by the author whilst working at SensePost Ltd[5] in South Africa called *reDuh* [36]- the tool allows an attacker who has already compromised a webserver to create a TCP circuit through the web server via a server side script he uploads. The client side of the tool makes regular polls to the script- passing encoded TCP sessions and commands. The attack could be identified by the numerous amounts of control messages being passed to the server in regular time intervals. An example of such a request is:

```
reDuh.jsp?command=newData&internalHost=intranet_db.company.com&port
    =3389&data=c29tZXJhd2JpbmFyeWRhdGE=
```

The **payload** content of TCP streams is of most interest to us. Discarding the payload (as many techniques do) may result in certain attacks not being identifiable. Many modern attacks have moved away from exploiting the protocol stack and instead focus on the application layer. For example, another tool developed by the author called *jmx_prod* exploits vulnerable JMX-Console implementations, uploading a command shell via an unusual HTTP request. SQL injection, Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF) and other such application layer attacks are not detectable via inspection of lower level layers. It is common to identify such attacks via signature based systems, but keeping these databases up to date is a challenging problem. Calculating several general metrics with the content of TCP flows may be useful in distinguishing these attacks.

3.3.8 Summary

In this chapter we discussed the datasets available for training and evaluating intrusion detection systems. We described the process by which features were extracted from the raw DARPA traffic and inserted into a database. The final database structure contains 142 columns and its structure is presented in the appendix. We present a sample row from the 2.8million present in the database in Table 3.13 listing all features and their values for a flow tagged as a warez attack.

The total number of files involved and time taken for each processing step is summarized in Table 3.12. In the next chapter we describe the classification process.

Table 3.12: Processing Summary

Task	Time Taken
Demultiplex Large Files	7 hours, 37 minutes
Header Statistics	20 hours, 40 minutes
Extract Flow Payloads	21 hours, 42 minutes
Analyze Payloads	2 hours, 50 minutes
Interpacket Statistics	18 hours, 21 minutes
Interflow Statistics	15 hours, 38 minutes
Import Attack Lists	25 minutes
Total	**3 days, 15 hours**

[5] www.sensepost.com

Table 3.13: Sample row from database

host_a	host_b	port_a	port_b	first_packet	last_packet	total_packets_a2b	total_packets_b2a
3279149676	**2886758594**	**15194**	**25**	**901299904**	**901299905**	**15**	**13**
resets_sent_a2b	resets_sent_b2a	ack_pkts_sent_a2b	ack_pkts_sent_b2a	pure_acks_sent_a2b	pure_acks_sent_b2a	sack_pkts_sent_a2b	sack_pkts_sent_b2a
0	**0**	**14**	**13**	**3**	**2**	**0**	**0**
dsack_pkts_sent_a2b	dsack_pkts_sent_b2a	max_sack_blks_ack_a2b	max_sack_blks_ack_b2a	unique_bytes_sent_a2b	unique_bytes_sent_b2a	actual_data_pkts_a2b	actual_data_pkts_b2a
0	**0**	**0**	**0**	**2951**	**374**	**10**	**9**
actual_data_bytes_a2b	actual_data_bytes_b2a	rexmt_data_pkts_a2b	rexmt_data_pkts_b2a	rexmt_data_bytes_a2b	rexmt_data_bytes_b2a	zwnd_probe_pkts_a2b	zwnd_probe_pkts_b2a
2951	**374**	**0**	**0**	**0**	**0**	**0**	**0**
zwnd_probe_bytes_a2b	zwnd_probe_bytes_b2a	outoforder_pkts_a2b	outoforder_pkts_b2a	pushed_data_pkts_a2b	pushed_data_pkts_b2a	SYN_pkts_sent_a2b	FIN_pkts_sent_a2b
0	**0**	**0**	**0**	**10**	**9**	**1**	**1**
SYN_pkts_sent_b2a	FIN_pkts_sent_b2a	req_1323_ws_a2b	req_1323_ts_a2b	req_1323_ws_b2a	req_1323_ts_b2a	adv_wind_scale_a2b	adv_wind_scale_b2a
1	**1**	**N**	**N**	**N**	**N**	**0**	**0**
req_sack_a2b	req_sack_b2a	sacks_sent_a2b	sacks_sent_b2a	urgent_data_pkts_a2b	urgent_data_pkts_b2a	urgent_data_bytes_a2b	urgent_data_bytes_b2a
N	**N**	**0**	**0**	**0**	**0**	**0**	**0**
mss_requested_a2b	mss_requested_b2a	max_segm_size_a2b	max_segm_size_b2a	min_segm_size_a2b	min_segm_size_b2a	avg_segm_size_a2b	avg_segm_size_b2a
1460	**1460**	**1460**	**92**	**6**	**19**	**295**	**41**
max_win_adv_a2b	max_win_adv_b2a	min_win_adv_a2b	min_win_adv_b2a	zero_win_adv_a2b	zero_win_adv_b2a	avg_win_adv_a2b	avg_win_adv_b2a
32120	**32736**	**512**	**32735**	**0**	**0**	**30012**	**32735**
initial_window_bytes_a2b	initial_window_bytes_b2a	initial_window_pkts_a2b	initial_window_pkts_b2a	ttl_stream_length_a2b	ttl_stream_length_b2a	missed_data_a2b	missed_data_b2a
23	**92**	**1**	**1**	**2951**	**374**	**0**	**0**
truncated_data_a2b	truncated_data_b2a	truncated_packets_a2b	truncated_packets_b2a	data_xmit_time_a2b	data_xmit_time_b2a	idletime_max_a2b	idletime_max_b2a
0	**0**	**0**	**0**	**0.05**	**0.16**	**1051.5**	**1040.4**
hardware_dups_a2b	hardware_dups_b2a	throughput_a2b	throughput_b2a	flow_duration	ip_t_mean_a2b	ip_t_variance_a2b	ip_t_stdev_a2b
0	**0**	**2432**	**308**	**2**	**0.09**	**0.08**	**0.28**
ip_p_mean_a2b	ip_p_variance_a2b	ip_p_stdev_a2b	ip_t_mean_b2a	ip_t_variance_b2a	ip_t_stdev_b2a	ip_p_mean_b2a	ip_p_variance_b2a
64.08	**694.58**	**26.35**	**0.1**	**0.09**	**0.3**	**232**	**192219**
ip_p_stdev_b2a	pl_total_chars_a2b	pl_unigram_ent_a2b	pl_bigram_ent_a2b	pl_most_freq_char_a2b	pl_count_mfc_a2b	pl_prop_mfc_a2b	pl_mf_bigram_a2b
438.43	**1000**	**5.19**	**7.9**	**101**	**78**	**0.08**	**105108**
pl_count_mf_bigram_a2b	pl_prop_mf_bigram_a2b	pl_num_of_zeroes_a2b	pl_prop_zeroes_a2b	pl_num_prnt_a2b	pl_num_nonprnt_a2b	pl_prop_prnt_a2b	pl_prop_nonprnt_a2b
21	**0.02**	**12**	**0.01**	**960**	**40**	**0.96**	**0.04**
pl_total_chars_b2a	pl_unigram_ent_b2a	pl_bigram_ent_b2a	pl_most_freq_char_b2a	pl_count_mfc_b2a	pl_prop_mfc_b2a	pl_mf_bigram_b2a	pl_count_mf_bigrm_b2a
375	**5.08**	**7.42**	**32**	**39**	**0.1**	**13010**	**9**
pl_prop_mf_bigrm_b2a	pl_num_of_zeroes_b2a	pl_prop_zeroes_b2a	pl_num_prnt_b2a	pl_num_nonprnt_b2a	pl_prop_prnt_b2a	pl_prop_nonprnt_b2a	total_conns_10s_anyport
0.02	**12**	**0.03**	**356**	**19**	**0.95**	**0.05**	**1**
total_conns_10s_sameSrvrPrt	total_packets_a2b_10s	total_packets_b2a_10s	attack_score	attack_name	attack_class		
0	**21**	**19**	**1**	**warez**	**warez**		

Chapter 4

Classification with Weka

In this chapter we discuss our classification process. We used binary classification to separate our data into "attack" and "normal" TCP flows and multiclass classification techniques to separate them into various attack classes or normal traffic (i.e. DoS, exploit, warez, probing or normal). We explain the subset of features we chose to evaluate on and describe the experiments we ran. All classifications were performed with the Weka framework, which is a collection of machine learning algorithms specifically designed for data mining tasks. We conclude the chapter with a description of methods of evaluating IDSs.

4.1 Sequential Minimal Optimization Algorithm

We used a SVM to classify our network traffic data. Specifically, we used the SMO algorithm implemented in Weka as discussed in Chapter 2, which implements a computationally efficient SVM. The options specified for the algorithm are presented in Table 4.1. We note that the exponent value of 1.0 (-E 1.0) reduces the polynomial kernel (PolyKernel) to linear classification.

We again stress that we treated the SVM as a 'blackbox', using the default configurations. Tuning the SVM algorithm is a very difficult process and there exists no optimal method to do so. The usual approach is to perform a grid search on all possible options. For example, the way to determine a useful complexity parameter c is simply to run classifications on a large range of possible values and choose the one which gives the best classification. As our research was more concerned with simply ascertaining the usefulness of our proposed features we chose to leave the algorithm's parameters in their default configuration.

4.2 Importing to Weka

We developed a tool called *export_to_arff.pl* to handle exporting data from the MySQL database to the Weka .arff file format. This tool performs two operations. Firstly, it creates configuration files which list the features to be used in extracting columns from the database. The user is prompted to enter desired columns from the database that he would like to include in the analysis (for example,

Table 4.1: SMO Algorithm Options

Option	Value	Description
c	1.0	The complexity parameter. Controls the trade off between the norm of a hyperplane and the separation accuracy [25].
kernel	PolyKernel -C 250007 -E 1.0	Kernel used for "kernel trick" which converts data into a higher space to aid in separation. See Section 2.4 for more details.
toleranceParameter	0.0010	Used to determine acceptable error margins when dealing with points that cannot easily be separated. See Section 2.5 for further information.

```
glenn@redtable:~$ perl export_to_arff.pl -table DARPA98 -generate_config_file config_new.cfg
New config file creation. Checking columns from week1_monday. Do you want to include:
host_a, varchar(15)? [Y\n]
host_b, varchar(15)? [Y\n]
port_a, int(11)? [Y\n]
first_packet, decimal(18,6)? [Y\n] n
last_packet, decimal(18,6)? [Y\n] n
total_packets_a2b, int(11)? [Y\n]
total_packets_b2a, int(11)? [Y\n]
....
total_packets_b2a_10s, int(11)? [Y\n]
attack_score, int(11)? [Y\n]
attack_name, varchar(30)? [Y\n]
attack_class, varchar(30)? [Y\n]
Writing "config_new.cfg" with 61 attributes. Done.

glenn@redtable:~$
```

Figure 4.1: export_to_arff.pl script

see Figure 4.1). Secondly, the tool uses a previously created configuration file to extract a sample from the database- the user specifies the type and number of attack samples desired with the *-fetch* flag. For example, the command:

```
perl export_to_arff.pl -table DARPA98 -fetch 'DoS,1500;probing,1500;
    exploit,1500;warez,1500;normal,6000' -c config_new.cfg -o
    weka_run_001
```

will fetch 1500 random samples of each attack class category and 6000 samples of normal traffic. The samples returned are from the columns specified in the *config_new.cfg* file. Internal logic converts the database variable types to Weka data types. The complete .arff Weka file will be written to *weka_run_001.arff*. Figure 4.2 depicts a sample .arff file with 8 features and the first 4 lines of data.

4.3 Feature Selection

We initially processed as many features as possible such that we could cut their number down at a later stage to a more efficient amount. This process is known as feature selection, and many complex techniques exist. It is an important step for any ML application. Several formal methods

```
% File generated by export_to_arff.pl @ Fri Aug 14 02:37:22 2009
% Data  consists of: (Normal,20)(warez,20)(probing,20)

@relation wekarun002

@attribute port_a,numeric
@attribute port_b,numeric
@attribute total_packets_a2b,numeric
@attribute total_packets_b2a,numeric
@attribute unique_bytes_sent_a2b,numeric
@attribute unique_bytes_sent_b2a,numeric
@attribute attack_score,{0,1}
@attribute attack_class,{Normal,warez,probing,DoS,exploit}

@data
15194,25,13,2951,374,1,warez
26134,25,12,1610339,1,warez
25618,21,33,26,234,766,0,Normal
17694,25,11,11,1024,329,0,Normal
```

Figure 4.2: Sample Weka .arff output file

are presented in papers such as *An Introduction to Variable and Feature Selection* [14] but we discovered the formal process of feature selection to be out of scope for this project. Furthermore, there exists no method which guarantees the best features to be returned. A common approach is to use forward or back selection in which different combinations are tried (adding and removing them sequentially).

We used a combination of intuition[1], Weka's *Select Attributes* tool and noting what other researchers in similar situations had used (such as Moore and Zuev's paper [5]) to determine which features to include. We separated features into different sets so that we could compare and contrast them. We attempted to have approximately the same number of features per class to facilitate fair testing. However, the number of *interflow* features is significantly smaller than the other feature classes. The processing time for calculating each interflow feature was significantly longer than those in other feature classes- for example, our table with 3 million rows (flows) would require 6 million SQL queries to SELECT, COUNT and UPDATE a single interflow feature. This resulted in 24 million queries for our 4 interflow features- a process which took some 15 hours to complete. We nonetheless included this feature set as we believe it has potential. Additionally, we broke the Header feature class into two separate classes of *HeaderFlag* and *HeaderByte*. The *HeaderFlag* set includes features relating to flags and options with in the TCP/IP headers. The *HeaderByte* set includes features relating to information about payload size and other data related statistics. The selection of attributes we chose are listed in Table 4.2

Within the HeaderFlag category, we chose to exclude source and destination IP addresses and ports. Whilst they are useful signs for identifying repeat offenders, we were more interested in identifying attacks by their local characteristics, as opposed to their source and destination.

4.4 Experiment Strategy

Our experiments were classed into two phases, and within each phase existed different categories. In **phase one**, our experiments evaluated different combinations of attack classes and feature sets. We

[1]Such as removing features known to be redundant, or those which appeared to always return the same value.

Table 4.2: Feature Sets for Classification

	Header-Flag	**Header-Byte**	**Payload**	**Interpacket**	**Interflow**
1.	resets_sent_a2b	total_packets_a2b	unigram_entropy_a2b	inter_packet_t_mean_a2b	flow_duration
2.	ack_pckts_sent_a2b	total_packets_b2a	bigram_entropy_a2b	inter_packet_t_variance_a2b	total_conns_10s_anyport
3.	pushed_data_pkts_a2b	unique_bytes_sent_a2b	most_freq_char_a2b	inter_packet_t_stdev_a2b	total_conns_10s_sameServerPort
4.	pushed_data_pkts_b2a	unique_bytes_sent_b2a	count_mfc_a2b	inter_packet_p_mean_a2b	total_packets_a2b_10s
5.	SYN_pkts_sent_a2b	actual_data_pkts_a2b	mf_bigram_a2b	inter_packet_p_variance_a2b	total_packets_b2a_10s
6.	FIN_pkts_sent_a2b	actual_data_pkts_b2a	count_mf_bigram_a2b	inter_packet_p_stdev_a2b	
7.	SYN_pkts_sent_b2a	rexmt_data_bytes_a2b	prop_zeroes_a2b	inter_packet_t_mean_b2a	
8.	FIN_pkts_sent_b2a	rexmt_data_bytes_b2a	prop_printable_a2b	inter_packet_t_variance_b2a	
9.	urgent_data_pkts_a2b	missed_data_a2b	prop_nonprintable_a2b	inter_packet_t_stdev_b2a	
10.	urgent_data_pkts_b2a	missed_data_b2a	unigram_entropy_b2a	inter_packet_p_mean_b2a	
11.	max_segm_size_a2b	truncated_packets_a2b	bigram_entropy_b2a	inter_packet_p_variance_b2a	
12.	max_segm_size_b2a	truncated_packets_b2a	most_freq_char_b2a	inter_packet_p_stdev_b2a	
13.	ttl_stream_length_a2b	idletime_max_a2b	count_mfc_b2a		
14.	ttl_stream_length_b2a	idletime_max_b2a	mf_bigram_b2a		
15.			count_mf_bigram_b2a		
16.			prop_zeroes_b2a		

Table 4.3: Experiment Options

Feature Sets	Attack Classes
HeaderFlags	Denial of Service (DoS)
HeaderBytes	Probing
Payload	Exploit
Interpacket	Warez
Interflow	

ran tests with both cross validation and separate training (seen attacks) and testing (unseen attacks) data. The purpose of this phase was to determine the best individual feature set or combination of feature sets. For these experiments we compared different evaluations by their Correctly Classified Instances, False Positive Rate and False Negative Rate (these terms are explained in Section 4.5).

We broke down our experiments in the first phase into different categories. We tested for various combinations between the sets of features and sets of attack classes (as per the columns in Table 4.3, e.g. FeatureSet{Payload} with AttackClass{DoS}, FeatureSet{Interpacket,Interflow} with AttackClass {DoS, Exploit, Warez, Probing}). We performed both binary and multiclass classification experiments. For the binary classification we separated samples into the categories "attack" and "normal". For multiclass classification, traffic was separated into the groups {DoS, Exploit, Warez, Probing, Normal}. Our goal was to evaluate the performance of the different feature sets against different combinations of attack samples, thus working out which feature set(s) performed the best at separating the data. We used the same sample sizes of attack and normal traffic for all experiments. When samples included multiple attacks, we used equal numbers of each attack class. We describe the experiments in further detail in Sections 4.4.1 and 4.4.2.

The purpose of our **second phase** of experiments was to compare our feature sets with results obtained by other researchers. For these experiments we used a training/testing approach similar to the other researchers. We compared our results with theirs by overlaying our Receiver Operating Characteristic (ROC) curve against theirs.

We made use of cross-validation for some of the experiment sets. With cross-validation our samples were partitioned into n sub-samples. Of these sub-samples we trained on *n-1* samples and tested on the remaining sample. This was then repeated n times (folds) resulting in every n sample being used once as validation data. The results from all the folds were averaged to give a single value. Initially we ran our experiments with 10 fold cross-validation which took a lengthy amount of time to run. Upon noticing no substantial difference in results with 4 fold cross-validation, we decided to use this number for all experiments. We used separate training and testing data for experiments involving the classification of previously unseen attacks.

4.4.1 Binary Classification Experiments

For the **first set of experiments**, we tested every combination of feature set and attack class, which yields 20 different combinations. The purpose of this set of experiments was to determine if different attack classes were best detected by certain features.

In the **second set of experiments**, we trained on data belonging to 3 out of 4 attack classes and tested on the 4th, yielding 20 different combinations. Therefore, in the testing stage, the classifier

Table 4.4: Experiment two, testing on unseen attacks

Trained On				Tested On	
DoS	probing	warez	normal	exploit	normal
exploit	probing	warez	normal	DoS	normal
exploit	DoS	warez	normal	probing	normal
exploit	DoS	probing	normal	warez	normal

(for all feature sets)

Table 4.5: Experiment three, all combinations of feature sets

ID	Feature Class Combinations	Number of Combinations
1	$\{S \subset F \mid \|S\| = 1\}$	5
2	$\{S \subset F \mid \|S\| = 2\}$	10
3	$\{S \subset F \mid \|S\| = 3\}$	10
4	$\{S \subset F \mid \|S\| = 4\}$	5
5	F	1

(tested against a sample containing all attacks)

had no previous examples of the attack class it was trying to classify. The purpose of this experiment was to examine how the SVM performed on classifying unseen attacks . This experiment setup is presented in Table 4.4.

In the **third set of experiments**, we tested every combination of feature classes on a set on the same set of data containing a sample of all attack classes, resulting in 31 different experiments. The purpose of this experiment set was to work out if certain combinations of attributes yield better results than individual combinations. For example, if the combination of interflow, interpacket and headerflags perform better than interpacket with payload. The list of combinations is described in Table 4.5.

In the **fourth set of experiments**, we reduced the number of features in every feature class to 5 features. This was done in order to match the number of features in the class with the fewest number of features- interflow having five. It was felt that this would give a better representation of the value of the interflow features as they did not have significant representation in earlier tests. As previously mentioned, the number of interflow features was kept low due to their long processing time. We ran the reduced feature sets against sample data containing equal number of all attack classes plus normal traffic. The reduced feature sets are presented in Table 4.6, in which the top 5 features from each category were kept (as evaluated by Weka's *Attribute Selection Tool*). The interflow category remained unchanged with *flow_duration,* total_conns_10s_anyport, total_conns_10s_sameServerPort, total_packets_a2b_10s, total_packets_b2a_10s. In a similar fashion to the third experiment above, we ran an experiment for every combination of the now reduced feature sets resulting in 31 different combination.

4.4.2 Multiclass Classification Experiments

For the multiclass experiment, we created a dataset with 1500 samples from each attack category and 1500 samples of normal traffic. The goal was to examine how well the SVM could separate traffic into the categories {DoS, probing, warez, exploit and normal} and to investigate if different feature sets result in attacks getting misclassified.

Table 4.6: Experiment four, equal size feature sets

HeaderFlag	HeaderByte	Interpacket	Payload
pushed_data_pkts_a2b	total_packets_a2b	packet_t_mean_a2b	unigram_entropy_a2b
max_segm_size_a2b	total_packets_b2a	packet_p_mean_a2b	bigram_entropy_a2b
ttl_stream_length_a2b	unique_bytes_a2b	packet_p_variance_a2b	num_of_zeroes_a2b
ttl_stream_length_b2a	unique_bytes_b2a	packet_t_mean_b2a	prop_zeroes_a2b
max_segm_size_b2a	actual_data_pkts_a2b	packet_p_mean_b2a	num_of_zeroes_b2a

4.4.3 Comparison of Results

Phase two was conducted in order to compare our results to those of Laskov and co-workers[25]. The authors of this paper tested various supervised and unsupervised algorithms including SVMs. We extracted a similar set of samples to those used this paper, and ran two similar experiments: classifying on *seen* attacks, and classifying on *unseen* attacks. We also compared our results to those of Zanero and Savaresi [31] who analysed the content of packets using unsupervised methods.

4.5 Evaluating Intrusion Detection Systems

There are several metrics used to evaluate IDSs. The most common include:

1. True Positives - Number of correctly identified attacks
2. False Positives - Number of non-attack instances identified as attack instances
3. True Negatives - Number of non-attack instances identified as non-attack
4. False Negatives - Number of instances of attack instances identified as non-attack

For binary classification, these values can be represented by a two-by-two confusion matrix. For example, if we are classifying between attack samples (A) and non-attack samples (N) we can depict the following confusion matrix:

A	N	
True Positives	False Positives	A
False Negatives	True Negatives	N

We can calculate several important metrics for intrusion detection; the Correctly Classified Instances (CCI), the Detection Rate (DR), the False Positive Rate (FPR) and the False Negative Rate (FNR):

$CCI = \frac{TP+TN}{A+N}$

$DR = \frac{TP}{TP+FN}$

$FPR = \frac{FP}{N}$

$FNR = \frac{FN}{A}$

The CCI is a measure of the total number of samples correctly identified. This is a standard metric for ML problems. However, with IDSs, we are more concerned with the DR, which measures how

many attacks are detected overall. The FPR is a useful indicator of how many alerts are in fact false, and the FNR an indicator of how many attacks were misidentified as non-attack. The DR is equivalent to recall from the field of Information Retrieval, whilst the false positive rate is similar to precision [31].

A trade off exists in IDS evaluation- as sensitivity is increased, more false positives are generated, but more instances are also correctly identified. We can represent this trade off with a ROC curve. A ROC curve has the FPR on the x-axis, and the TPR on the y-axis (both ranging from 0 to 1.0). We depict the ROC space in Figure 4.3. Several important points exist in the ROC space [12]: the lower left point of (0,0) represents never issuing a false positive, but then no true positives are identified either. The opposite approach would be represented at position (1,1), where every attack instance is caught, but with a 100% FPR. Point D on the graph (0,1) represented perfect classification, as every attack instance is caught with no false positives. Generally speaking, one point on a ROC curve is better than another if it is higher and to the left. The diagonal (y=x) represents random selection- i.e. classification at point C is no better than randomly guessing which class the instance belongs to (it seems to be guessing positive category 70% of the time). A classifier which randomly selects classes is expected to produce a point on the line y=x. Any classification appearing under the y=x line (such as point E) is worse than a random classifier. Inverting such a classifier makes it useful (as it would appear that it does have information about the classification, but is applying it incorrectly). A classifier which produces a single confusion matrix corresponds to a single point in the ROC space. A ROC curve can conceptually be generated by evaluating numerous outcomes in which the sensitivity of the classifier is adjusted and plotting the corresponding points.

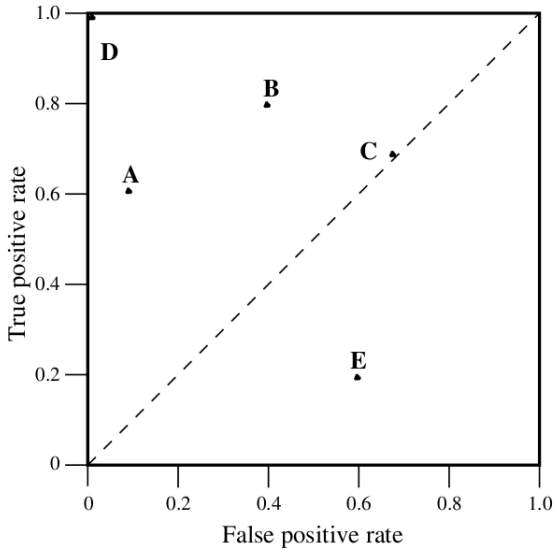


Figure 4.3: ROC Space [12]

For multiclass classification, a confusion matrix is also possible. As an example of classifying the attacks A1 to A5, we present the confusion matrix:

A1	A2	A3	A4	A5	
99.7%		0.3%			A1
	96.9%	0.6%	2.0%	0.5%	A2
	36.3%	**63.5%**	0.2%		A3
0.1%	60.0%	0.8%	**39.1%**		A4
	1.9%	15.6%	0.6%	**81.9%**	A5

We can read that the class A1 is classified as class A1 99.7% of the time and classified as A3 0.3% of the time (similarly for the other classes, where each row sums to 100%). In a confusion matrix, we generally hope to see large numbers in the diagonal from top left to bottom right, such as those in bold in the above example. The overall evaluation score of a multiclass classifier can be calculated as the average of this diagonal.

Chapter 5

Results

The results of running the classification experiments proposed in Chapter 4 are presented and discussed in this chapter. In the first phase of experiments, we explored our feature sets' ability to classify hostile traffic. Our goal was to ascertain which feature set or combination of feature sets gave the best results. In the second phase of experiments, we compared our results to those obtained from other researchers who have done similar work.

5.1 Phase One - Feature Evaluation

In the first phase of experiments, we ran binary and multiclass classifications. Through these experiments, we aimed to gain an understanding of the relationship between our proposed feature sets and the different attack classes. For the outcome of the binary classification experiments we present results for Correctly Classified Instances (%), False Positives (%), False Negatives (%) and Detection Rate (%). We chose to compare results based on the CCI %. For the multiclass experiments results are presented in confusion matrices.

5.1.1 Binary Classification

Binary classification aims to separate samples into two groups. These experiments aimed to separate TCP traffic samples into the groups of "attack" or "normal".

The results for our **first set of experiments** are presented in Table 5.1. This experiment was the easiest for the SVM algorithm to classify as each run contained only one attack type. The results indicate that different attacks are not best detected by different feature sets, and that our payload feature set (with one exception) is the best option for detecting all the classes of attack. We note that in the DoS category all the feature sets except for interflow achieved a correctly classified percentage greater than 99%, although interflow still performed well at 93%. DoS connections are easily identifiable as they are usually numerous, small (containing only few identical packets) and symmetrical (same header information and content in every connection). These characteristics would be evident in all the feature sets.

The results for the **second set of experiments** are presented in Table 5.2. We again observe the **payload** feature set yielding the highest correctly classified percentage in three out of four tests with correctly classified scores of 96.5%, 97.3% and 82.6%. The payload feature set was behind interpacket at 78.3% and HeaderFlags at 77.3% when training on exploit, DoS and probe traffic and testing on warez (with 72% correct classification). This experiment showed that the SVM performs worse when classifying unseen attacks, but not substantially so.

The results of the **third set** of experiments are presented in Tables 5.3, 5.4, 5.5 and 5.6. For each combination of 1,2,3,4 and 5 feature sets, the winning combination always included the payload feature set. The highest correctly classified score was unsurprisingly obtained by the combination of all five feature sets, with a score of 98.2%. However, the pair combination of payload and interpacket were only marginally behind with a score of 98.1%. The lowest scores were obtained with combinations containing the HeaderBytes feature set.

The results of the individual feature sets (now with 5 features each) from the **fourth experiment set** is presented in Table 5.7. It is interesting to note that the performance of the feature set increases with the reduction in size (comparing to Table 5.3). The headerbyte feature set decreases slightly in performance by 1.3%, payload by 9.1% and interpacket by 23.2%. The interflow feature set now performs on par with the headerflag, and better than the headerbytes and interpacket feature sets. This result indicates that calculating more interflow features could prove useful for future work. As mentioned in Section 4.3 we only calculated 5 features for the interflow set due to the long processing time with our offline data. For a real world scenario this bottleneck would not exsit, which may make the interflow feature set useful. In Tables 5.8, 5.9 and 5.10 we present the results of all the combinations of feature sets. Throughout these results we once again note the recurring theme of the payload feature set giving the best performance with results.

5.1.2 Multiclass Classification

The next set of experiments involved classifying TCP traffic into one of the categories of DoS, probing, warez, exploit or normal. The results of these tests are presented in the confusion matrices in Tables 5.11 through 5.15. Note that a blank entry denotes a score of 0.5% or below.

For the multiclass classification using the **headerflags** feature set (Table 5.11) we note that DoS was classified correctly over 99% of the time. A good result was obtained identifying exploit traffic with a correctly classified score of almost 97%. The majority of the probing traffic was misclassified as exploit, as was a large segment of the warez traffic. The overall score for this feature set was 76.24%, ranking it as second place overall.

The **headerbyte** feature set (Table 5.12) obtains a nearly perfect separation of exploit traffic at 99% but performs poorly on every other category. We further notice that the majority of all traffic (87%) was classified as exploit with this feature set. This feature set had the worst overall performance at 29.1%.

The **interpacket** feature set (Table 5.13) performed well on classifying DoS and warez samples with correctly classified scores of 99.6% and 97.8% respectively. The normal traffic classification was respectable with a correctly classified score of 80.1%. The classification of probing samples was poor with the majority of its samples being identified as exploit, warez or normal. The exploit

Table 5.1: Results of Experiment 1, Every Attack vs Every Feature Set

Features	Header - Flags	x					x				
	Header - Bytes		x					x			
	Payload			**x**					**x**		
	Inter-packet				x					x	
	Inter-Flow					x					x
Attacks	Exploits	x	x	x	x	x					
	DoS						x	x	x	x	x
	Probing										
	Warez										
Results	CCI (%)	97.7	50.9	**98.4**	97.8	70.8	99.7	99.2	**99.8**	99.6	93.0
	False Positive (%)	1.2	24.1	0.4	1.3	28.5	0.1	0.1	0.0	0.2	5.5
	False Negative (%)	1.1	25.0	1.1	0.9	0.7	0.1	0.7	0.1	0.2	1.5
	Detection Rate (%)	97.8	50.9	97.8	98.2	96.7	99.7	98.7	99.7	99.5	96.7

(i) Exploit vs all feature sets and (ii) DoS vs all feature sets

Features	Header - Flags	**x**					x				
	Header - Bytes		x					x			
	Payload			x					**x**		
	Inter-packet				x					x	
	Inter-Flow					x					x
Attacks	Exploits										
	DoS										
	Probing	x	x	x	x	x					
	Warez						x	x	x	x	x
Results	CCI (%)	**98.7**	66.7	98.0	88.7	58.4	91.3	65.8	**97.3**	78.2	67.1
	False Positive (%)	1.0	27.0	1.4	6.7	41.5	8.7	0.2	2.6	6.1	30.6
	False Negative (%)	0.4	6.4	0.6	4.6	0.1	0.0	34.0	0.1	15.6	2.3
	Detection Rate (%)	99.3	78.3	98.8	90.4	98.8	100.0	59.4	99.8	73.7	89.3

(iii) Probing vs all feature sets and (iv) Warez vs all feature sets

Table 5.2: Results of Experiment 2, Rotated Training on 3, Testing on 1

		Train: D,P,W. Test: E					Train: E,P,W. Test: D				
Features	Header - Flags	x					x				
	Header - Bytes		x					x			
	Payload			**x**					**x**		
	Inter-packet				x					x	
	Inter-Flow					x					x
Results	CCI (%)	79.1	50.1	**96.5**	94.8	50.0	44.3	51.1	**97.3**	47.1	10.8
	False Positive (%)	20.2	0.2	2.8	4.5	0.2	5.8	0.1	2.7	3.0	42.0
	False Negative (%)	0.7	49.7	0.6	0.7	49.8	49.9	48.7	0.0	49.9	47.2
	Detection Rate (%)	97.8	50.1	98.7	98.6	50.0	47.0	50.6	99.9	48.5	14.5

(a) Testing on Exploit and Denial of Service
(Datasets: Trained of 3x1500 attacks + 4500 normal; tested on 1x1500 attack + 1500 normal)

		Train: E,D,W. Test: P					Train: E,D,P. Test: W				
Features	Header - Flags	x					**x**				
	Header - Bytes		x					x			
	Payload			**x**					x		
	Inter-packet				x					x	
	Inter-Flow					x					x
Results	CCI (%)	64.3	49.8	**82.6**	79.8	52.4	**77.3**	67.6	72.0	78.3	68.2
	False Positive (%)	19.9	0.2	2.2	4.3	1.0	22.6	0.2	1.4	3.4	0.5
	False Negative (%)	15.8	50.0	15.2	15.9	46.6	0.1	32.2	26.5	18.3	31.3
	Detection Rate (%)	65.6	49.9	75.9	74.1	51.3	99.6	60.7	64.7	71.8	61.3

(b) Testing on Probing and Warez Traffic
(Datasets: Trained of 3x1500 attacks + 4500 normal; tested on 1x1500 attack + 1500 normal)

Table 5.3: Results of Experiment 3.1, Individual Features

Features	Header - Flags	x				
	Header - Bytes		x			
	Payload			**x**		
	Inter-packet				x	
	Inter-Flow					x
Results	CCI (%)	67.3	54.5	**97.1**	87.2	66.1
	False Positive (%)	21.6	0.2	2.5	3.7	0.2
	False Negative (%)	11.1	45.4	0.3	9.1	33.8
	Detection Rate (%)	71.9	52.3	99.3	83.6	59.6

(Dataset: 1500 of each attack class, 6000 normal)

Table 5.4: Results of Experiment 3.2, Pair Combinations

Features	Header - Flags	x	x	x	x						
	Header - Bytes	x				x	x	x			
	Payload		**x**			**x**			**x**	**x**	
	Inter-packet			x			x		x		x
	Inter-Flow				x			x		x	x
Results	CCI (%)	67.4	**97.8**	90.4	69.2	**97.4**	90.8	63.8	**98.1**	**97.4**	88.7
	False Positive (%)	21.5	1.8	3.7	21.8	2.4	3.3	0.2	1.7	2.3	3.2
	False Negative (%)	11.1	0.4	5.9	9.1	0.3	5.9	36.1	0.3	0.3	8.0
	Detection Rate (%)	72.0	99.2	88.7	75.7	99.4	88.8	58.0	99.5	99.4	85.4

(Dataset: 1500 of each attack class, 6000 normal)

Table 5.5: Results of Experiment 3.3, Triple Combinations

Features	Header - Flags	x	x	x	x	x	x				
	Header - Bytes	x			x		x	x	x	x	
	Payload	**x**	**x**			**x**		**x**		**x**	**x**
	Inter-packet		x	x	x			x	x		x
	Inter-Flow			x		x	x		x	x	x
Results	CCI (%)	97.8	**98.1**	90.6	90.6	**97.8**	69.3	98.1	**90.6**	**97.4**	98.1
	False Positive (%)	1.8	1.6	3.5	3.6	1.8	21.7	1.6	3.2	2.4	1.7
	False Negative (%)	0.4	0.3	5.8	5.8	0.4	9.0	0.3	6.2	0.3	0.2
	Detection Rate (%)	99.2	99.5	88.8	88.8	99.2	75.9	99.5	88.3	99.4	99.5

(Dataset: 1500 of each attack class, 6000 normal)

Table 5.6: Results of Experiment 3.4 Quadruple Combinations, 3.5 All Five Feature Sets

Features	Header - Flags		x	x	x	x	x
	Header - Bytes	x		x	x	x	x
	Payload	**x**	**x**		**x**	**x**	x
	Inter-packet	x	x	x		x	x
	Inter-Flow	x	x	x	x		x
Results	CCI (%)	98.1	**98.2**	90.5	97.8	**98.1**	**98.2**
	False Positive (%)	1.6	1.6	3.5	1.8	1.6	1.6
	False Negative (%)	0.2	0.3	6.0	0.4	0.3	0.2
	Detection Rate (%)	99.5	99.5	88.5	99.2	99.5	99.5

(Dataset: 1500 of each attack class, 6000 normal)

Table 5.7: Results of Experiment 4.1, Individual Features

Features	Header - Flags (5)	x				
	Header - Bytes (5)		x			
	Payload (5)			**x**		
	Inter-packet (5)				x	
	Inter-Flow (5)					x
Results	CCI (%)	68.3	53.2	**88.1**	64.1	66.1
	False Positive (%)	22.1	0.1	2.5	33.1	0.2
	False Negative (%)	9.6	46.8	9.4	2.8	33.8
	Detection Rate (%)	74.3	51.6	83.5	85.8	59.6

(Dataset: 1500 of each attack class, 6000 normal)

Table 5.8: Results of Experiment 4.2, Pair Combinations

Features	Header - Flags (5)	x	x	x	x						
	Header - Bytes (5)	x				x	x	x			
	Payload (5)		**x**			**x**			**x**	**x**	
	Inter-packet (5)			x			x		x		x
	Inter-Flow (5)				x			x		x	x
Results	CCI (%)	70.6	**91.9**	89.4	74.1	89.3	61.1	63.6	88.4	89.4	63.4
	False Positive (%)	22.0	2.9	4.8	22.5	2.5	4.7	0.1	2.6	2.6	4.7
	False Negative (%)	7.4	5.2	5.8	0.3	8.2	34.3	36.3	8.0	8.0	31.9
	Detection Rate (%)	79.2	90.1	88.6	98.8	85.3	57.0	57.9	84.1	85.5	58.7

(Dataset: 1500 of each attack class, 6000 normal)

Table 5.9: Results of Experiment 4.3, Triple Combinations

Features	Header - Flags	x	x	x	x	x	x				
	Header - Bytes	x			x		x	x	x	x	
	Payload	**x**	**x**			**x**		**x**		**x**	**x**
	Inter-packet		x	x	x			x	x		x
	Inter-Flow			x		x	x		x	x	x
Results	CCI (%)	93.5	**92.4**	89.4	89.3	91.8	74.5	90.1	65.7	91.1	89.6
	False Positive (%)	2.9	3.0	4.8	4.7	3.1	22.4	2.6	1.9	2.6	2.6
	False Negative (%)	3.5	4.6	5.8	6.0	5.2	3.1	7.3	32.4	6.3	7.8
	Detection Rate (%)	93.0	91.0	88.6	88.3	90.1	89.9	86.7	59.8	88.3	85.9

(Dataset: 1500 of each attack class, 6000 normal)

Table 5.10: Results of Experiment 4.4 Quadruple Combinations, 4.5 All Five Feature Sets

Features	Header - Flags		x	x	x	x	x
	Header - Bytes	x		x	x	x	x
	Payload	**x**	**x**		**x**	**x**	x
	Inter-packet	x	x	x		x	x
	Inter-Flow	x	x	x	x		x
Results	CCI (%)	91.3	**92.3**	89.3	93.4	94.0	94.1
	False Positive (%)	2.7	3.05	4.6	3.1	3.0	2.9
	False Negative (%)	6.1	4.69	6.0	3.5	3.0	3.0
	Detection Rate (%)	88.7	90.9	88.3	93.0	94.0	93.8

(Dataset: 1500 of each attack class, 6000 normal)

classification was also poor, with the majority of its samples being classified as warez. The overall interpacket score was 63.69%, ranking it as 3rd place.

The **interflow** feature set (Table 5.14) performed well for the classification of DoS and warez samples, obtaining scores of 94.7% and 99.5% for these attacks respectively. However, the classification of exploit and probing samples were 4.2% and 16.0% respectively. The classification of normal traffic was even worse obtaining a score of 0%. These 3 categories had the majority of their traffic classified as warez traffic. The high volume of traffic with DoS and warez traffic (both number of connections and amount of data transferred) may explain the success of the interflow feature set in obtaining the high classification success rate with these attacks. The overall interflow correct classification score was 42.97%, ranking it as 4th place.

The **payload** feature set (Table 5.15) once again had the best classification result. For every attack category a score of over 97% was achieved. The classification of normal traffic was the lowest score with 92.3%.

Only the payload feature set achieved a high success rate for classifying every attack. Every other feature set largely misclassified at least one attack class.

5.2 Phase Two - Comparison to Other Research

Phase two of our experiment set was conducted in order to compare our results to those of Laskov and co-workers [25], and Zanero and Savaresi [31]. Lasko and co-workers compared various supervised and unsupervised algorithms for classifying the DARPA data sets, including the SVM

Table 5.11: Confusion Matrix - Headers-Flag Feature Set

(7500 Samples, 76.24% Correct Classification)

DoS	exploit	warez	probing	normal	
99.7%					DoS
	96.9%	0.6%	2.0%		exploit
	36.3%	**63.5%**			warez
	60.0%	0.8%	**39.1%**		probing
	1.9%	15.6%	0.6%	**81.9%**	normal

Table 5.12: Confusion Matrix - Headers-Byte Feature Set

(7500 Samples, 29.1% Correct Classification)

DoS	exploit	warez	probing	normal	
25.5%	74.4%				DoS
	99.9%				exploit
	99.1%				warez
9.9%	67.5%	2.9%	**19.7%**		probing
4.5%	94.9%				normal

Table 5.13: Confusion Matrix - Interpacket Feature Set

(7500 Samples, 63.69% Correct Classification)

DoS	exploit	warez	probing	normal	
99.6%					DoS
	36.7%	58.8%		4.4%	exploit
	1.1%	**97.8%**		0.9%	warez
1.3%	25.5%	36.8%	**4.3%**	32.2%	probing
2.7%	14.8%	1.9%		**80.1%**	normal

Table 5.14: Confusion Matrix - Interflow Feature Set

(7500 Samples, 42.9% Correct Classification)

DoS	exploit	warez	probing	normal	
94.7%		3.0%	2.2%		DoS
	4.2%	93.2%	2.5%		exploit
		99.5%			warez
	25.1%	58.5%	**16.4%**		probing
13.0%		86.7%			normal

Table 5.15: Confusion Matrix - Payload Feature Set

(7500 Samples, 97.39% Correct Classification)

DoS	exploit	warez	probing	normal	
99.7%					DoS
	97.7%	0.8%	0.6%	0.9%	exploit
		97.5%	1.0%	1.3%	warez
			99.7%		probing
	2.3%		4.9%	**92.3%**	normal

Table 5.16: Results of Experiment 5, Testing on Unseen Attacks from DARPA

Features	Header - Flags	x				
	Header - Bytes		x			
	Payload			**x**		
	Inter-packet				x	
	Inter-Flow					x
Results	CCI (%)	68.3	59.7	80.2	79.0	70.5
	False Positive (%)	18	1.3	1.3	5.5	0.8
	False Negative (%)	13.7	39.0	18.5	15.5	28.7
	Detection Rate (%)	70.0	55.5	72.5	74.2	63.2

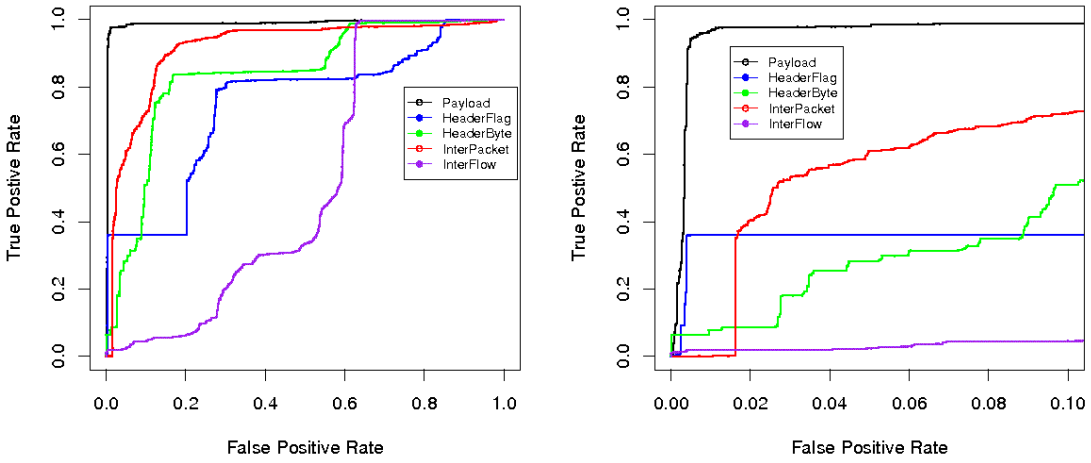
algorithm. They did not process the raw DARPA traffic but instead made use of the already extracted features supplied for the Knowledge Discovery and Data Mining 1999 (KDD'99) contest [1]. There are 41 features[1] in the KDD'99 list which are broken up into 3 sections: basic features of individual TCP connections (e.g protocol type), content features within a connection created from specific domain knowledge (e.g is_guest_login) and traffic features calculated over a two second window[2] (e.g number of connections to the host). There are no features calculated automatically from the payload. Lasko and co-workers performed two experiments which were of interest to us: classification with an SVM on *seen* attacks, and classification with an SVM on *unseen* attacks. Our previous experiment of various attacks matched their *seen* one, but their experiment involving unseen attacks differed from ours conducted in Section 4.4.1. They trained on the official training data and then tested on the two weeks of official testing data, whereas we combined both sets and tested on unseen attack classes. We therefore extracted a similar sample to theirs and ran the classification experiment. Their paper only presents ROC graphs (without figures) so we performed ROC analysis in Weka on our data and exported the results to be graphed in R[3]. The results of this new experiment of training on the official training data and testing on the official testing data (i.e testing on unseen attacks) is presented in Table 5.16. The ROC curves for this experiment and the previously conducted *seen* attacks experiment are presented in Figures 5.1 and 5.2. As expected we observe the common theme of the payload feature set giving the best performance. Visually comparing these results to the ROC curve obtained in Lasko and co-workers' paper it, was evident that the our payload feature set could possibly be performing better, but that our other feature sets had performed poorly in comparison. In Figure 5.3, we overlay the ROC curves from our payload feature set with the ROC curves from the paper by Lasko and co-workers (consisting of various algorithms, including the SVM). In (a) we compare the curves for *seen* attacks and in (b) *unseen* attacks. For the *seen* case we notice that there is little room for improvement, and our results overlap theirs peaking very quickly to $> 99\%$ with a fractional false positive rate. However, in the *unseen* case we see that our payload feature set performs somewhat better than their classification, even with our feature set having less than half the number of features. Additionally, using only 5 features from our payload feature set, we equal the performance of the KDD'99 features with false positive rates greater than 0.05%.

The detection rates and false positives rates from Zanero and Savaresi [31] are presented in column 1 and 2 of Table 5.17. In their research, they made use of SOMs to cluster the DARPA traffic,

[1] See Appendix B for the complete list of KDD'99 features

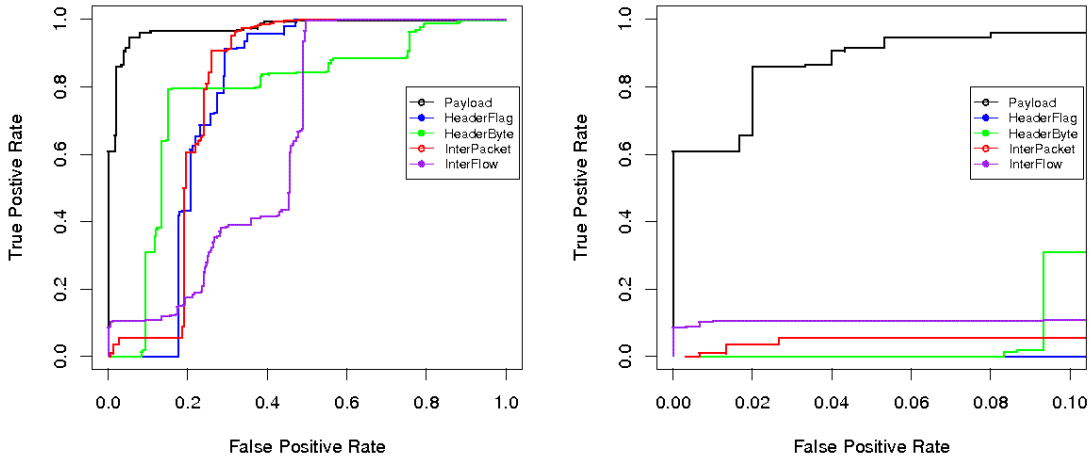
[2] We discovered the KDD'99 feature set after completing our classification exercise- it was interesting to see similar features to those we calculated in our interflow feature set.

[3] www.r-project.org



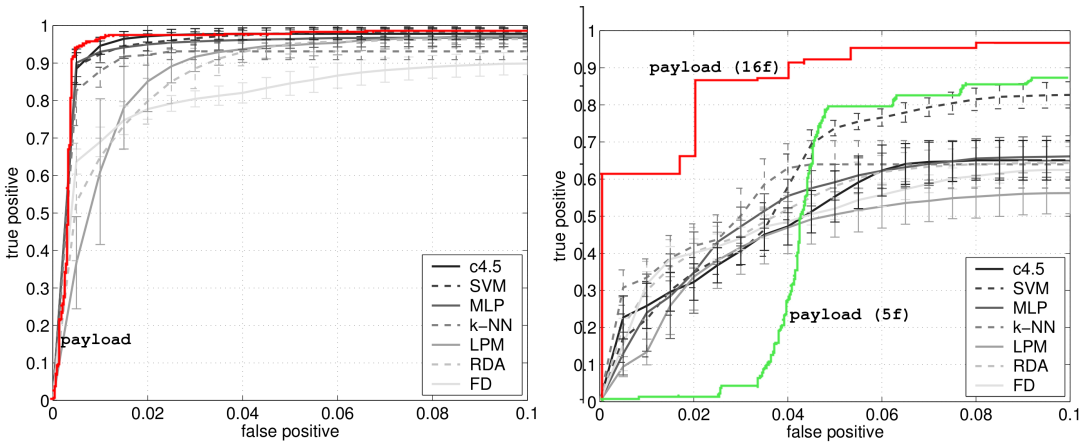
(a) x-axis to 1.0 (b) x-axis to 0.1

Figure 5.1: ROC Curve for *Seen* Attacks



(a) x-axis to 1.0 (b) x-axis to 0.1

Figure 5.2: ROC Curves for *Unseen* Attacks



(a) Tested on *Seen* Attacks (b) Tested on *Unseen* Attacks

Figure 5.3: ROC Curve from [25] with our payload result superimposed

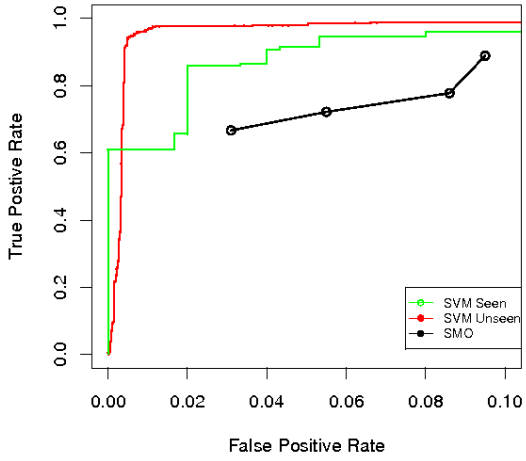


Figure 5.4: Our SVM vs SMO from [31]

focusing on the the payload content of individual packets. We include the results from our SVM classifier using payload features along side for comparison and plot the results on an ROC curve in Figure 5.4. Whilst supervised and unsupervised approaches are not directly comparable, this similarity of results further highlights the usefulness of our simple payload features.

5.3 Results Summary

In the first phase of our experiment setup, we compared and contrasted various combinations of feature sets and attack classes. We can conclude that different attacks are not best detected by certain feature sets as the payload feature set gave the best results for detecting all attacks. When testing on previously unseen attacks, the payload feature set performed well, whilst the other feature sets showed a large drop in performance. The best performance was obtained by using all 5 feature

Table 5.17: False Positive Rate vs Detection Rate for SMO [31] and Our SVM (payload)

False Positive Rate	SMO DR	Our SVM DR on *Seen*	Our SVM DR on *Unseen*
0.031%	66.7%	97.83%	86.7%
0.055%	72.2%	98.63	94.66%
0.086%	77.8%	98.88%	96.0%
0.095%	88.9%	98.9%	96.0%

sets, but the score for the combination of payload and interpacket was only 0.1% behind. The poor performance of the interflow feature set in initial experiments can be explained by its comparatively small number of features (see Table 4.2). Upon limiting all feature sets to their 'best' 5 features (see Table 4.6), we observed that the interflow set performed on par with the headerflags and interpacket sets, and better than the headerbyte set. However, the payload feature set performed better with 5 features than any of the other feature sets did with the 5 feature or full feature experiments.

When classifying traffic into one of the categories of normal, DoS, exploit, warez or probe the payload feature set once again gave the best performance. It was the only feature set to not largely misclassify at least one class of traffic.

Comparing our results to other research in the field we note that the header, interflow and interpacket features give worse performance in comparison to their results, but our payload feature set give better performance (see ROC curves in Figures 5.3 and 5.4). The results of Zanero's work is not directly comparable to ours but the similarity of results does emphasise the utility of payload inspection techniques.

In conclusion of our results, we note that the attributes chosen for the feature sets of headerflags, headerbytes, interpacket and interflow did not perform well in separating the DARPA network traffic as compared to:

(i) our payload feature set or
(ii) methods proposed by other researchers. [25, 31].

In all of our experiments, the payload feature set performed well, obtaining substantially better scores as compared to our other feature sets, out-performing the results from Lasko and co-workers [25] and obtaining comparable results to Zanero and Savaresi [31].

Chapter 6

Conclusion

Our goal in this project was to ascertain the utility of different features of network traffic in classifying individual TCP flows as either hostile or benign. Of particular interest to us were the attributes we calculated from inspecting the content of TCP flows. We also calculated 3 other sets of features; header based, interpacket based and interflow based. Our particular interest in the payload stems from the fact that the majority of existing IDSs discard the payload (with more emphasis on features similar to the other 3 we used), but we hypothesized that significant value could be derived from the simple analysis of them.

The results of our experiments indicated that our feature sets for interpacket, interflow and header values did not perform well compared to other research. However, the results from our payload inspection features were promising: compared to our other feature sets the payload set performed substantially better. Our payload features also performed better compared to other methods which used a combination of basic features of TCP connections, features calculated over a sliding window and specific domain knowledge features [25].

Payload inspection is a difficult task in a real time IDS due to the rate at which traffic passes through the device. However, as other authors [37] have shown it is entirely feasible on modern hardware. We therefore believe that our features could be calculated on a real time IDS, especially since our payload features are sufficiently simple so as not to require much processing time.

Although our results were positive, we are concerned about the contrived nature of the DARPA datasets (as was discussed in Section 3.1.1). They are out of date (not containing any modern attacks) and are not representative of real world traffic (as they were artificially generated). Because of this there is a concern that ours and others' classification techniques based on the data could not detect real world attacks, but merely detect the irregularities in the contrived DARPA traffic. It would be interesting to see how our system performs on other datasets, such as the recently released *2009 Inter-Service Academy Cyber Defense Competition* labeled data sets [9]. Unfortunately this data was released near the end of our research and so we were not able to make use of it. As Dan Weber, one of the engineers involved in the creation of the DARPA datasets noted on the release of this new dataset, "You wouldn't try to use Windows 95A today. Don't use ancient datasets."

In conclusion, we now know that simple content analysis of TCP traffic can be extremely useful when attempting to identify hostile TCP traffic. This research represents a confluence of results,

correlating with other recent work [6, 31, 37]. With many modern attacks only being distinguishable from regular traffic in the application layer, content analysis will be very useful for new IDS designs.

Bibliography

[1] KDD Cup 1999. Online, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, [Accessed July 2009].

[2] Support vector machine. Online, http://en.wikipedia.org/wiki/Support_Vector_Machine, [Accessed July 2009].

[3] The 1998 intrusion detection off-line evaluation plan. Technical report, MIT Lincoln Laboratory, Information Systems Technology Group, 1998.

[4] Introduction to support vector machine (SVM) models. Online, http://www.dtreg.com/svm.htm, [Accessed July 2009].

[5] D. Zuev A. Moore. Internet traffic classification using bayesian analysis techniques. *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, 2005.

[6] D. Zuev A. Moore, M. Crogan. Discriminators for use in flow-based classification. Technical report, Queen Mary University of London, 2005.

[7] K. Papagiannaki A. Moore. Toward the accurate identification of network applications. *Passive and Active Network Measurement*, pages 41–54, 2005.

[8] J. Anderson. Computer security threat monitoring and surveillance. Technical report, Washington, Pennsylvania, 1980.

[9] T. Cook R. Fanelli E. Dean W. Adams C. Morrell G. Conti B. Sangster, T. O'Connor. Toward instrumenting network warfare competitions to generate labeled datasets. Technical report, United States Military Academy, West Point, New York, 2009.

[10] T. Brugger. KDD cup '99 dataset considered harmful. Technical report, UC Davis, Department of Computer Science.

[11] S. Jajodia N. Wu D. Barbara, J. Couto. ADAM: a testbed for exploring the use of data mining in intrusion detection. *SIGMOD Rec.*, 30(4):15–24, 2001.

[12] T. Fawcett. An introduction to ROC analysis. Technical report, Institute for the Study of Learning and Expertise, 2005.

[13] Tristan Fletcher. Support vector machines explained. Technical report, UCL, 2009.

[14] A. Elisseeff I. Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3, 2003.

[15] J. Bedell I. Korf, M. Yandell. *BLAST*. O'Reilly, 2003.

[16] K. Ingham. *Anomaly Detection for HTTP Intrusion Detection: Algorithm Comparisons and the Effect of Generalization on Accuracy*. PhD thesis, Department of Computer Science, University of New Mexico, 2007.

[17] W. Chu S. Shevade A. Poo K. Duan, S. Keerthi. Multi-category classification by soft-max combination of binary classifiers. In *In 4th International Workshop on Multiple Classifier Systems*, 2003.

[18] R. Vemuri K. Labib. NSOM: A real-time network-based intrusion detection system using self-organizing maps. Technical report, Department of Applied Science, University of California, Davis.

[19] C. Kreibich. Netdude: The hacker's choice. Online, http://netdude.sourceforge.net/, [Accessed July 2009].

[20] L. Rozonoer M. Aizerman, E. Braverman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[21] P. Chan M. Mahoney. Detecting novel attacks by identifying anomalous network packet headers. Technical report, 2001.

[22] P. Chan M. Mahoney. An analysis of the 1999 DARPA/lincoln laboratory evaluation data for network anomaly detection. In *In Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection*, pages 220–237. Springer-Verlag, 2003.

[23] P. Chan M. Mahoney. Learning rules for anomaly detection of hostile network traffic. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 601, Washington, DC, USA, 2003. IEEE Computer Society.

[24] S. Ostermann. tcptrace. Online, http://jarok.cs.ohiou.edu/software/tcptrace/, [Accessed July 2009].

[25] C. Schafer K. Rieck P. Laskov, P. Dussel. Learning intrusion detection: Supervised or unsupervised? In *Image Analysis and Processing*, 2005.

[26] M. Heywood P. Piotr. Dynamic intrusion detection using self-organizing maps. Technical report, Faculty of Comp. Science, Dalhousie University, 2002.

[27] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998.

[28] J. Postel. RFC 791: Internet Protocol, 1981.

[29] J. Postel. RFC 793: Transmission Control Protocol, 1981.

[30] J. Reynolds and J. Postel. RFC 1700: Assigned Numbers. October 1994.

[31] S. Savaresi S. Zanero. Unsupervised learning techniques for an intrusion detection system. Technical report, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2004.

[32] S. Tanenbaum. *Computer networks: 2nd edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[33] J. Tevessen. tcpflow - a TCP flow recorder. Online, http://www.circlemud.org/ jelson/software/tcpflow/, [Accessed July 2009].

[34] V. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[35] S. Stolfo W. Lee. Data mining approaches for intrusion detection. In *7th USENIX Security Symposium*, 1998.

[36] G. Wilkinson. reDuh: Re-inventing TCP like its 1973! Online, http://www.sensepost.com/research/reDuh/, [Accessed July 2009].

[37] S. Zanero. Analyzing TCP traffic patterns using self organizing maps. In *ICIAP '05: Proc. 13th International Conference on Image Analysis and Processing, volume 3617 of LNCS*, pages 8–8. SpringerVerlag, 2005.

Appendix A

SQL Queries

The following is the MySQL command used in *flow_parser.pl* to create the main table.

```
CREATE TABLE DARPA(
host_a                  VARCHAR(15),
host_b                  VARCHAR(15),
port_a                  INT,
port_b                  INT,
first_packet            DECIMAL(18,6),
last_packet             DECIMAL(18,6),
total_packets_a2b       INT,
total_packets_b2a      INT,
resets_sent_a2b            INT,
resets_sent_b2a            INT,
ack_pkts_sent_a2b  INT,
ack_pkts_sent_b2a  INT,
pure_acks_sent_a2b INT,
pure_acks_sent_b2a INT,
sack_pkts_sent_a2b INT,
sack_pkts_sent_b2a INT,
dsack_pkts_sent_a2b        INT,
dsack_pkts_sent_b2a        INT,
max_sack_blks_ack_a2b      INT,
max_sack_blks_ack_b2a      INT,
unique_bytes_sent_a2b      INT,
unique_bytes_sent_b2a      INT,
actual_data_pkts_a2b       INT,
actual_data_pkts_b2a       INT,
actual_data_bytes_a2b      INT,
actual_data_bytes_b2a      INT,
rexmt_data_pkts_a2b        INT,
rexmt_data_pkts_b2a        INT,
rexmt_data_bytes_a2b       INT,
rexmt_data_bytes_b2a       INT,
zwnd_probe_pkts_a2b        INT,
```

```
zwnd_probe_pkts_b2a        INT,
zwnd_probe_bytes_a2b       INT,
zwnd_probe_bytes_b2a       INT,
outoforder_pkts_a2b        INT,
outoforder_pkts_b2a        INT,
pushed_data_pkts_a2b       INT,
pushed_data_pkts_b2a       INT,
SYN_pkts_sent_a2b  INT /*Preprocessed */,
FIN_pkts_sent_a2b        INT,
SYN_pkts_sent_b2a  INT,
FIN_pkts_sent_b2a  INT,
req_1323_ws_a2b            CHAR(1) /*Preprocessed*/,
req_1323_ts_a2b               CHAR(1),
req_1323_ws_b2a                CHAR(1),
req_1323_ts_b2a                CHAR(1),
adv_wind_scale_a2b     INT,
adv_wind_scale_b2a INT,
req_sack_a2b               CHAR(1),
req_sack_b2a           CHAR(1),
sacks_sent_a2b         INT,
sacks_sent_b2a             INT,
urgent_data_pkts_a2b       INT,
urgent_data_pkts_b2a       INT,
urgent_data_bytes_a2b      INT,
urgent_data_bytes_b2a      INT,
mss_requested_a2b  INT,
mss_requested_b2a  INT,
max_segm_size_a2b  INT,
max_segm_size_b2a  INT,
min_segm_size_a2b  INT,
min_segm_size_b2a  INT,
avg_segm_size_a2b  INT,
avg_segm_size_b2a  INT,
max_win_adv_a2b            INT,
max_win_adv_b2a            INT,
min_win_adv_a2b            INT,
min_win_adv_b2a            INT,
zero_win_adv_a2b   INT,
zero_win_adv_b2a   INT,
avg_win_adv_a2b            INT,
avg_win_adv_b2a            INT,
initial_window_bytes_a2b INT,
initial_window_bytes_b2a INT,
initial_window_pkts_a2b        INT,
initial_window_pkts_b2a    INT,
ttl_stream_length_a2b      INT,
ttl_stream_length_b2a      INT,
missed_data_a2b            INT,
```

```
missed_data_b2a            INT,
truncated_data_a2b INT,
truncated_data_b2a INT,
truncated_packets_a2b      INT,
truncated_packets_b2a      INT,
data_xmit_time_a2b FLOAT,
data_xmit_time_b2a       FLOAT,
idletime_max_a2b FLOAT,
idletime_max_b2a FLOAT,
hardware_dups_a2b        INT,
hardware_dups_b2a  INT,
throughput_a2b             INT,
throughput_b2a             INT,
inter_packet_t_mean_a2b         FLOAT,
inter_packet_t_variance_a2b      FLOAT,
inter_packet_t_stdev_a2b FLOAT,
inter_packet_p_mean_a2b          FLOAT,
inter_packet_p_variance_a2b      FLOAT,
inter_packet_p_stdev_a2b FLOAT,
inter_packet_t_mean_b2a          FLOAT,
inter_packet_t_variance_b2a      FLOAT,
inter_packet_t_stdev_b2a FLOAT,
inter_packet_p_mean_b2a          FLOAT,
inter_packet_p_variance_b2a      FLOAT,
inter_packet_p_stdev_b2a FLOAT,
payload_total_chars_a2b         INT,
payload_unigram_entropy_a2b        FLOAT,
payload_bigram_entropy_a2b       FLOAT,
payload_most_freq_char_a2b       INT,
payload_count_mfc_a2b              INT,
payload_prop_mfc_a2b               FLOAT,
payload_mf_bigram_a2b            INT,
payload_count_mf_bigram_a2b        INT,
payload_prop_mf_bigram_a2b FLOAT,
payload_num_of_zeroes_a2b        INT,
payload_prop_zeroes_a2b            FLOAT,
payload_num_printable_a2b        INT,
payload_num_nonprintable_a2b       INT,
payload_prop_printable_a2b FLOAT,
payload_prop_nonprintable_a2b    FLOAT,
payload_total_chars_b2a          INT,
payload_unigram_entropy_b2a        FLOAT,
payload_bigram_entropy_b2a       FLOAT,
payload_most_freq_char_b2a       INT,
payload_count_mfc_b2a              INT,
payload_prop_mfc_b2a               FLOAT,
payload_mf_bigram_b2a            INT,
payload_count_mf_bigram_b2a        INT,
```

```
payload_prop_mf_bigram_b2a FLOAT,
payload_num_of_zeroes_b2a        INT,
payload_prop_zeroes_b2a            FLOAT,
payload_num_printable_b2a        INT,
payload_num_nonprintable_b2a       INT,
payload_prop_printable_b2a FLOAT,
payload_prop_nonprintable_b2a    FLOAT,
total_conns_10s_anyport         INT,
total_conns_10s_sameServerPort     INT,
total_packets_a2b_10s              INT,
total_packets_b2a_10s              INT,
attack_score    INT,
attack_name        VARCHAR(100),
attack_class VARCHAR(100)
);
```

The following is the MySQL command used in *hostileTrafficImporter.pl* to create tables for the supplied truth tables.

```
CREATE TABLE $truth_table (
date VARCHAR(20),
duration VARCHAR(20),
service VARCHAR(10),
tcp_srcPort INT,
tcp_dstPort INT,
ip_src VARCHAR(15),
ip_dst VARCHAR(15),
attack_score INT,
attack_name VARCHAR(100)
);
```

The following are the commands used in flow_parser.pl and hostileTrafficImporter.pl to create indexes. The column first_packet is almost 99% unique.

```
CREATE INDEX ix_darpa ON DARPA (first_packet,host_a,host_b,port_a,port_b);
CREATE INDEX ix_truthtable_$t ON $t (date,ip_src,ip_dst,tcp_srcPort,tcp_dstPort);
```

Appendix B

KDD'99 Traffic Features

In Section 5.2 we compared our system to that of Laskov and co-workers [25]. The authors of this paper made use of the features from the KDD'99 cup. The features are as follows, as taken from the KDD'99 website [1].

Table1: Basic Features for individual TCP Sessions

Feature Name	Description
duration	length (number of seconds) of the connection
protocol_type	type of the protocol, e.g. tcp, udp, etc.
service	network service on the destination, e.g., http, telnet, etc.
src_bytes	number of data bytes from source to destination
dst_bytes	number of data bytes from destination to source
flag	normal or error status of the connection
land	1 if connection is from/to the same host/port; 0 otherwise
wrong_fragment	number of "wrong" fragments
urgent	number of urgent packets

Table 2: Content features within a connection suggested by domain knowledge.

Feature Name	Description
hot	number of "hot" indicators
num_failed_logins	number of failed login attempts
logged_in	1 if successfully logged in; 0 otherwise
num_compromised	number of "compromised" conditions
root_shell	1 if root shell is obtained; 0 otherwise
su_attempted	1 if "su root" command attempted; 0 otherwise
num_root	number of "root" accesses
num_file_creations	number of file creation operations
num_shells	number of shell prompts
num_access_files	number of operations on access control files
num_outbound_cmds	number of outbound commands in an ftp session
is_hot_login	1 if the login belongs to the "hot" list; 0 otherwise
is_guest_login	1 if the login is a "guest"login; 0 otherwise

Table 3: Traffic features computed using a two-second time window.

Feature Name	Description
count	number of connections to the same host as the current connection in the past two seconds
serror_rate	% of connections that have "SYN" errors
rerror_rate	% of connections that have "REJ" errors
same_srv_rate	% of connections to the same service
diff_srv_rate	% of connections to different services
srv_count	number of connections to the same service as the current connection in the past two seconds
srv_serror_rate	% of connections that have "SYN" errors
srv_rerror_rate	% of connections that have "REJ" errors
srv_diff_host_rate	% of connections to different hosts

Appendix C

Source Code

C.1 flow_parser.pl

```
#!/usr/bin/perl -w
use strict;
use File::Find;
use DBI;
use Net::TcpDumpLog;
use NetPacket::IP qw(:strip);
use NetPacket::TCP qw(:strip);
use Statistics::Descriptive;
use File::Slurp;                        #Quick file reading
use Data::Dumper;

## flow_parser.pl, takes a list of files as output from tcptrace+tcpflow
## and performs various inspection operations.

##Precons:
#1. Large dump file (e.g DARPAs 500MB type)
#2. demux the dump file with lndtool -r Demux. This creates folders + subfolders for
     each flow
#3. Apply tcpflow:
#       find . | grep ".trace"| xargs -n 50 tcptrace -lrn --csv > ./tcptrace.output2

select(STDERR); $| = 1;         # make unbuffered
select(STDOUT); $| = 1;         # make unbuffered

#DB variables
my $database="DBI:mysql:project";
my $username="user";
my $password="12d2ded7bba8008cc176cc6e7ff2d619";
my $dbh_g;              #global db handle

print "##This tool does various traffic inspection tasks. No args for usage.\n";
if ($#ARGV+1 <1){
        print "ERROR: Not enough arguments\n";
        &usage();
        exit;
}
```

```
sub usage{
        print "GLOBAL OPTIONS:\n";
        print " -dir = specifies the directory in which to search for dump/trace
            files.\n\n";
        print " -dump = Search for .tcpdump/.dump files in the -dir specified folder
             and demux
        them into individual sessions. For each dump file found a folder called
            demux_<dump file name>
        is created. Under this directory each session file is created.\n\n";
        print "-tbl <tbl_name> = Populate <tbl_namet> with stats generated by -trace
             [-rtt] (header), or AMMEND payload
        statistic (see -analyze_payloads) options. That is to say you must have
            already done a -trace on the output table option,
        or have included -trace in your command.\n\n";
        print "-tbl_as_filename = Checks to see which demux folder each .trace file
            is inside and puts it
        into a table reflecting that (for -trace and -analyze_payloads). For example
            , all trace files inside
        /foo/demux_sample/ (and it's subdirectories) will be placed in a table
            called \"sample\". This option
        pretty much only works with output created by the -dump option.\n";
        print "-darpa_to_sql = If you have downloaded the DARPA IDS datasets with
            our other tools they will be
        in the form: <year>/<Training|Testing>/week<n>/<monday|tuesday|wednesday>
            thursday|friday>/<tcpdumpfiles(s)>

        After running the -dump option each <tcpdumpfile> will have a demux folder
            containing
        all of its sessions. Including the -darpa_to_sql flag makes use of this
            structure and will
        process folders looking for week<n>/<day> names, creating SQL tables of the
            same form. Process
        training/test folders individually. This is best used with the -tbl_prefix
            and -tbl_postfix
        options:
        -tbl_prefix = Table prefix used when creating multiple tables with -
            darpa_to_sql
        -tbl_postifx =Table prefix used when creating multiple tables with -
            darpa_to_sql

        An example of using the darpa option is below:

        perl flow_parser.pl -dir /home/user/DARPA/1998/Training/ -dump -trace -
            darpa_to_sql -tbl_prefix tbl_1998_tr_ -tbl_postfix _tcptrace
        Explanation:    1. This will first demux all tcpdump files into individual
            sessions (-dump)
                        2. All the session files will have stats calculated
                        3. Each session stats will be inseted into a table of the
                             form tbl_1998_tr_week1_tuesday_tcptrace
                           (the name is generated from the post/pre-fix options, and
                                the directory structure)

        An example of applying the tool to a non DARPA type folder set:

        perl flow_parser.pl -dir ./foo/ -dump -trace -tbl my_stats_table  <-- Puts
```

```
            stats from all dump files into the same table
        perl flow_parser.pl -dir ./foo/ -dump -trace -tbl_as_filename     <-- Puts
            stats for each dump file into its own table (table name=dump file name)\
            n";

        print "\nFLOW HEADER OPTIONS:\n";
        print "-trace = Search for .trace files (created by -dump) and extract
            statistics about ttraffic concerning
        packet sizes, headers etc. 92 fields genereated for each flow. If -trace
            option is not paired with -trace_to_sql
        .csv output goes to STDOUT\n\n";
        print "-rtt = Used in conjunction with -trace. Will include an extra stats
            including RTT, bringing the total to 142 stats\n\n";

        print "\nINTER PACKET OPTIONS:\n";
        print "-inter_packet_stats = Calculates several statistics regarding packet
            behaviour within a flow\n\n";

        print "FLOW CONTENT OPTIONS:
        -extract_payloads = Generate flow payload for all trace files.
        -analyze_payloads = Inspects the payload contents for all trace files and
            calculates:
                        -Entropy
                        -Number of zeroes
                        -Most frequent character
        Specify -tbl <table_name> for SQL output (we will create column names in
            that table, and match (timestamp,ipsrc,ipdst,portsrc,portdst).
        If -tbl option is not present, output will be sent to STDOUT in .CSV format.
             This option of course depends on -extract_payloads either
        being included in the command, or having been previously run.

        \n";
}

my $dir="";
#my $DARPA_table_prefix="tbl_98_tr_";
#my $DARPA_table_postfix="_tcptrace";
my ($tbl_prefix,$tbl_postfix)=("","");
my ($dump,$trace,$flow_payload)=(0,0,0);
my $use_darpa_dirs_to_sql=-1;
my $rtt=0;              #Denotes if we include tcptrace's RTT flag, ie extra output
my $output_table="";
my $tbl_as_filename=0;
my $operation_number=1; #For pretty output of 1. Doing xxx, 2. Doing yyy
my $display_counter=0;  #For progress printing
my %table_list_tcptrace;        #index to keep track of which tables/cols have been
    created,
my %column_list_tcptrace;       # useful to prevent multiple SQL lookups
my $tcpflow_maxbytes=1000;      #Maximum number of bytes for each sessions payload
    of reconstruction.
my $analyze_payloads=0;
my $calc_IAT=0;
#my $payload_analysis_to_db=0;  #CL option for payload analysis, will create new
    columns in $table and then populate them with the results. Otherwise .csv to
    stdout
#my $drop_if_exist=1;           #Used for the analyze_payload subroutine. If set to
```

```
    1 it'll drop the calculated columns in the table if they exist (ie entropy
    column etc)

for(my $i=0; $i<$#ARGV+1;$i++){

        if($ARGV[$i] eq "-dir"){        #dump to many demuxed trace files
                $dir=$ARGV[$i+1];$i++;
        }
        elsif($ARGV[$i] eq "-dump"){    #apply tcptrace to all .trace files
                $dump=1;
        }
        elsif($ARGV[$i] eq "-trace"){   #apply tcptrace to all .trace files
                $trace=1;
        }
        elsif($ARGV[$i] eq "-inter_packet_stats"){
                $calc_IAT=1;
        }
        elsif($ARGV[$i] eq "-extract_payloads"){        #apply tcpflow to all .trace
             files
                $flow_payload=1;
        }
        elsif($ARGV[$i] eq "-rtt"){     #include RTT stats with -trace
                $rtt=1;
        }
        elsif($ARGV[$i] eq "-tbl_prefix"){      #
                $tbl_prefix=$ARGV[$i+1];$i++;
        }
        elsif($ARGV[$i] eq "-tbl_postfix"){     #
                $tbl_postfix=$ARGV[$i+1];$i++;
        }
        elsif($ARGV[$i] eq "-tbl"){             #Output to a single table
                $output_table=$ARGV[$i+1];$i++;
                if($output_table eq ""){
                        print "ERROR: No table name given. Exiting.";
                }
#               $trace=1;
        }
        elsif($ARGV[$i] eq "-darpa_to_sql"){    #
                $use_darpa_dirs_to_sql=1;
        }
        elsif($ARGV[$i] eq "-tbl_as_filename"){ #
                $tbl_as_filename=1;
#               $trace=1;
        }
        elsif($ARGV[$i] eq "-analyze_payloads"){        #
                $analyze_payloads=1;
        }
        else{
                print "ERROR: Unknown argument: $ARGV[$i]. Exiting.\n";
                exit;
        }
#       elsif($ARGV[$i] eq "-payload_anaylsis_to_sql"){ #
#               $payload_analysis_to_db=1;
#       }

}
```

```
##Todo: apply logic to which options are allowed in what combinations.
if($dir eq ""){print "ERROR: You didn't  give me a direcotry.\n"; exit;}
if(! -e $dir){print "ERROR: $dir not found.\n"; exit;}
##

##Open DB connections, if we'll be performing DB tasks:
if($use_darpa_dirs_to_sql == 1 || $output_table ne "" || $tbl_as_filename == 1 ){
        print "\n## ".$operation_number++ .". Opening database connection. ";
        $dbh_g = DBI->connect($database, $username, $password) || die "Could not
            connect to database: $DBI::errstr";
        print "Done.\n";
}


if($dump == 1){
        print "## ".$operation_number++ .". Demuxing dump files\n";
        find(\&demux_dumps, $dir);
        print "# Done\n";
        #print "Done demultiplexing dump files into individual sessions.\n";
}

# Apply tcptrace and/or IAT_stats to .trace files
if($trace == 1 || $calc_IAT == 1){
        if($trace ==1 && $calc_IAT ==1){
                print "\n## ".$operation_number++ .". Applying tcptrace/IAT to all
                    trace files\n";
        }
        elsif($trace == 1 && $calc_IAT==0){
                print "\n## ".$operation_number++ .". Applying tcptrace to all trace
                     files\n";
        }
        elsif($trace==0 && $calc_IAT==1){
                print "\n## ".$operation_number++ .". Calculating interpacket
                    statistics for all sessions (trace files)\n";
        }
        $display_counter=0;
        find(\&find_trace_files, $dir);
        print "\n# Done\n";
}

if($flow_payload==1){
        #1. Extract the payload of each .trace file. Each trace file will generate
            files of the form:
        #     163.001.236.180.47248-209.085.227.099.00080
        #     srcIP.port-dstIP.port
        print "\n## ".$operation_number++ .". Extracting payload contents of all .
            trace files\n";
        $display_counter=0;
        find(\&tcpflow_it,$dir);
        print"\n# Done\n";

}

if($analyze_payloads==1){
        print "\n## ".$operation_number++ .". Taking a look at the generated payload
```

```
            files.\n";
        $display_counter=0;
        find(\&analyze_payload_content, $dir);
        print "\n# Done\n";
}



######
# Last thing we do is close the DB if we were using it. Make sure this code stays at
     the end
##########
if($use_darpa_dirs_to_sql == 1 || $output_table ne "" || $tbl_as_filename == 1){
        print "\n## ".$operation_number++ .". Closing database connection. ";
        $dbh_g->disconnect();
        print"Done.\n";
}


########################################
##Subroutines called by find follow      #
########################################
sub create_index{
        my $table=shift;
        my @columns=@_;
        print "Creating index ";
}

sub find_trace_files{
        /\.trace$/ or return;                   #a. only trace files
        if(/non_ip.trace/){return;}             #b. except the non_ip file
        my $filename=$_;
        my $directory=$File::Find::dir;
        if($File::Find::dir=~m/\/p(\d+)\//){    #c. And only TCP
                my $protocol_number=$1;
                if($protocol_number != 6){
                        $File::Find::prune=1;
                        return;
                }

        }
        else{
                print "ERROR: Cannot determine protocol number in use.. Exiting.\n";
                print "Offending file: $File::Find::name\n\n";
                exit;
        }

        if(++$display_counter % 50 == 0){
                print "$display_counter trace files poked\r";
        }

        #Must do tcptrace first to create the primary table
        if($trace == 1){
                my $tmp_table=get_trace_tablename($directory);
                &tcptrace_it($filename,$tmp_table);
        }
```

```
        #and then can add stats
        if($calc_IAT == 1){
                my $tmp_table=get_trace_tablename($directory);
                &do_inter_packet_stats($filename,$tmp_table);
        }


}

#Pass a directory, return the table name to use for output
sub get_trace_tablename{

                        my $dirname=shift;
                        my $tmpTableName="";

                        #2.1 If DARPA structure
                        if($use_darpa_dirs_to_sql == 1){
                                if($dirname=~m/(week\d)\/(monday|tuesday|wednesday|
                                    thursday|friday)\//){
                                        $tmpTableName=$tbl_prefix . $1 . "_" . $2 .
                                            $tbl_postfix;
                                }
                                else{
                                        print "##ERROR: DARPA folders not in
                                            required format of week<n>/<day>\n.";
                                        exit;
                                }
                        }
                        #2.2 Else if outputing all trace files to one SQL table
                        elsif($output_table ne ""){
                                $tmpTableName = $tbl_prefix . $output_table .
                                    $tbl_postfix;
                        }
                        #2.3 Else if outputing to one table per filename
                        elsif($tbl_as_filename == 1){
                                if($dirname=~m/demux_(.+?)\//){
                                        $tmpTableName = $tbl_prefix . $1 .
                                            $tbl_postfix;

                                }
                                else{
                                        print "\n## Error extracting dump filename
                                            that the .trace files belong to. I look
                                            for the pattern demux_<filename>, but
                                            couldn't find it.\n";
                                        print "## The trace file I'm looking at is
                                            $File::Find::name and the name I
                                            extracted was \"$tmpTableName\"\n";
                                        print "## Please check the usage for the -
                                            trace -tbl_as_fname option. The demux_<
                                            filename> needs to first be created by
                                            the -dump option.";
                                        exit;
                                }
                        }
                        #2.4 Error
```

```
                        else{
                                print "ERROR: It seems I wan't provided an output
                                    type- DARPQ SQL, one table, or one table per
                                    dump file\n";
                                exit;
                        }

                        if(!exists $table_list_tcptrace{$tmpTableName} ){
                                if($trace == 1){
                                        #Create the table
                                        &create_tcptrace_table($tmpTableName);
                                }
                                else{
                                        if(&check_table_existence($tmpTableName) ==
                                            0){
                                                print "Cannot find table
                                                    $tmpTableName - have you run the
                                                     -trace option first?\n";
                                                exit;
                                        }
                                }
                        $table_list_tcptrace{$tmpTableName}=1;
                        }

                return $tmpTableName;

}

sub analyze_payload_content{
        #1. Open each payload file (generated by tcpflow) and do some analysis
        my $filename=$_;
        my $dir=$File::Find::dir;

        /\d+\.\d+-\d{3}.\d{3}.\d{3}.\d{3}.\d{5}-\d{3}.\d{3}.\d{3}.\d{3}.\d{5}/ or
            return;
        if($dir=~m/\/p(\d+)\//){        #c. And only TCP
                my $protocol_number=$1;
                if($protocol_number != 6){
                        #print "Not TCP traffic, ignoring ($protocol_number)\n";
                        $File::Find::prune=1;
                        return;
                }
        }
        else{
                print "ERROR: Cannot determine protocol number in use.. Exiting.\n";
                print "Offending file: $File::Find::name\n\n";
                exit;
        }

        if(++$display_counter % 50 == 0){
                print "$display_counter payloads analyzed\r";
        }

        my $timestamp=substr($filename,0, index($filename,'-')-1);
        #1.1 Convert <epoch.msecs>-163.001.236.180.47248-209.085.227.099.00080 to
            163.1.236.180.47248-209.85.227.99.80
```

```
        my ($ip_src, $ip_dst, $tcp_srcPort, $tcp_dstPort) = &remove_leading_zeroes(
            substr($filename, index($filename,'-')+1));

#       my ($entropy,$most_freq_char, $count_of_mfc, $num_zeroes)=&
    file_content_inspector($filename);
#       print "$filename :: H=$entropy,MFC=$most_freq_char, MFC_count=$count_of_mfc,
     Zeroes=$num_zeroes\n";


        my ($total_chars,$H_unigram,$most_freq_char,$count_of_mfc,$proportion_mfc,
            $num_zeroes,$proportion_zeroes,$num_of_ascii_printable,
            $num_of_ascii_non_printable,$prop_printable,$prop_non_printable,
            $H_bigram,$most_freq_bigram,$count_of_mf_bi,$proportion_mf_bigram)=&
            file_content_inspector($filename);

        #We need to figure out which direction the flow is in. We get that from the
            directory structure:
        # e.g /S163.1.236.180/D129.67.1.207/ indicates that 163.x is the source, and
             129.x the destination
        $dir=~m/S(\d+\.\d+\.\d+\.\d+)\/D(\d+\.\d+\.\d+\.\d+)/;
        my $traffic_originator=$1;
        my $traffic_destination=$2;

        #3. Work out output table name
        my $tmpTableName=get_trace_tablename($dir);

        my $sql_ins_pl="";
        if($traffic_originator eq $ip_src){
                $sql_ins_pl="UPDATE $tmpTableName SET payload_total_chars_a2b=
                    $total_chars, payload_unigram_entropy_a2b=$H_unigram,
                    payload_bigram_entropy_a2b=$H_bigram, payload_mf_bigram_a2b=
                    $most_freq_bigram, payload_count_mf_bigram_a2b=$count_of_mf_bi,
                    payload_prop_mf_bigram_a2b=$proportion_mf_bigram,
                    payload_most_freq_char_a2b=$most_freq_char,
                    payload_count_mfc_a2b=$count_of_mfc, payload_prop_mfc_a2b=
                    $proportion_mfc, payload_num_of_zeroes_a2b=$num_zeroes,
                    payload_prop_zeroes_a2b=$proportion_zeroes,
                    payload_num_printable_a2b=$num_of_ascii_printable,
                    payload_num_nonprintable_a2b=$num_of_ascii_non_printable,
                    payload_prop_printable_a2b=$prop_printable,
                    payload_prop_nonprintable_a2b=$prop_non_printable WHERE host_a='
                    $ip_src' AND host_b='$ip_dst' AND port_a='$tcp_srcPort' AND
                    port_b='$tcp_dstPort' AND first_packet>$timestamp-30 AND
                    first_packet<$timestamp+30";
                #$sql_ins_pl="UPDATE $tmpTableName SET payload_entropy_a2b=$entropy,
                     payload_most_freq_char_a2b=$most_freq_char,
                    payload_num_of_zeroes_a2b=$num_zeroes WHERE host_a='$ip_src' AND
                     host_b='$ip_dst' AND port_a='$tcp_srcPort' AND port_b='
                    $tcp_dstPort' AND first_packet>$timestamp-10 AND first_packet<
                    $timestamp+10";
        }
        elsif($traffic_destination eq $ip_src){
                $sql_ins_pl="UPDATE $tmpTableName SET payload_total_chars_b2a=
                    $total_chars, payload_unigram_entropy_b2a=$H_unigram,
                    payload_bigram_entropy_b2a=$H_bigram, payload_mf_bigram_b2a=
                    $most_freq_bigram, payload_count_mf_bigram_b2a=$count_of_mf_bi,
```

```
                    payload_prop_mf_bigram_b2a=$proportion_mf_bigram,
                    payload_most_freq_char_b2a=$most_freq_char,
                    payload_count_mfc_b2a=$count_of_mfc, payload_prop_mfc_b2a=
                    $proportion_mfc, payload_num_of_zeroes_b2a=$num_zeroes,
                    payload_prop_zeroes_b2a=$proportion_zeroes,
                    payload_num_printable_b2a=$num_of_ascii_printable,
                    payload_num_nonprintable_b2a=$num_of_ascii_non_printable,
                    payload_prop_printable_b2a=$prop_printable,
                    payload_prop_nonprintable_b2a=$prop_non_printable WHERE host_b='
                    $ip_src' AND host_a='$ip_dst' AND port_b='$tcp_srcPort' AND
                    port_a='$tcp_dstPort' AND first_packet>$timestamp-30 AND
                    first_packet<$timestamp+30";
                #$sql_ins_pl="UPDATE $tmpTableName SET payload_entropy_b2a=$entropy,
                     payload_most_freq_char_b2a=$most_freq_char,
                    payload_num_of_zeroes_b2a=$num_zeroes WHERE host_b='$ip_src' AND
                     host_a='$ip_dst' AND port_b='$tcp_srcPort' AND port_a='
                    $tcp_dstPort' AND first_packet>$timestamp-10 AND first_packet<
                    $timestamp+10";
        }
        else{
                print "ERROR: Unable to determine source and destination IPs of
                    traffic flow from directory structure.\n";
                exit;
        }
        my $sth_payload_stats = $dbh_g->prepare($sql_ins_pl);
        $sth_payload_stats->execute or die "SQL Error: $DBI::errstr\nFull SQL
            Command was $sql_ins_pl\n";
        if($sth_payload_stats->rows == 0){
                #print "WARNING: 0 rows were updated whilst doing analysis for
                    $ip_src:$ip_dst at $timestamp\n";
        }
}

sub demux_dumps{
        /\.dump|tcpdump$/ or return;                            # only operate on
            dump files
        my $filename=substr($_,0,rindex($_,'.'));       #remove .dump file extension
        #print "Demuxing $File::Find::name\n";
        print "#  File: $File::Find::name \t => $File::Find::dir/demux_$filename \n"
            ;
        system("lndtool -r Demux -M -p -o demux_$filename $_");# 2>>./lndtool.err");
}

sub tcptrace_it{
        my $file=shift;
        my $tmp_table=shift;

        if(! -e $file || $tmp_table eq ""){
                print "ERROR: Bad args passed to tcptrace_it subroutine.\n";
                exit;
        }

#       if(++$display_counter % 50 == 0){
#               print "$display_counter files traced\r";
#       }
```

```
my $conns=0;
my $tcptracecommand="tcptrace --csv -ln";
if($rtt){
        $tcptracecommand="tcptrace --csv -lnr";
}
#1. Get the output
my @tcptraceoutput=`$tcptracecommand $file`;
for(my $i=0; $i<@tcptraceoutput;$i++){
        #print $tcptraceoutput[8] . "\n";
        if($tcptraceoutput[$i]=~m/^conn/)
        {
                #Two lines after the heading (conn_#,host_a,host_b,etc)
                    comes a line of the data
                #We remove whitespace, split Y/Y and 1/2 into Y,Y and 1,2
                    and then encase in single quotes ready for SQL
                my $csdata = $tcptraceoutput[$i+2];
                $csdata=~s/\s//g;       #Remove white space
                $csdata=~s/\//,/g;      #Replace Y/Y and 1/2 etc with Y,Y
                    and 1,2 (ie for SYN/FIN_pkts_sent_a2b etc, splitting SYN
                     and FIN into their own fields)
                $csdata=~s/NA/-1/g;
                $csdata= join(',',map {"'$_'"} split(',', substr($csdata,
                    index($csdata,",")+1)));


                #1.2. Do we want to include RTT stats?
                #2. Build the INSERT query
                my $sql_insert="";
                if($rtt){
                        $sql_insert="INSERT INTO $tmp_table(host_a, host_b,
                            port_a, port_b, first_packet, last_packet,
                            total_packets_a2b, total_packets_b2a,
                            resets_sent_a2b, resets_sent_b2a,
                            ack_pkts_sent_a2b, ack_pkts_sent_b2a,
                            pure_acks_sent_a2b, pure_acks_sent_b2a,
                            sack_pkts_sent_a2b, sack_pkts_sent_b2a,
                            dsack_pkts_sent_a2b, dsack_pkts_sent_b2a,
                            max_sack_blks_ack_a2b, max_sack_blks_ack_b2a,
                            unique_bytes_sent_a2b, unique_bytes_sent_b2a,
                            actual_data_pkts_a2b, actual_data_pkts_b2a,
                            actual_data_bytes_a2b, actual_data_bytes_b2a,
                            rexmt_data_pkts_a2b, rexmt_data_pkts_b2a,
                            rexmt_data_bytes_a2b, rexmt_data_bytes_b2a,
                            zwnd_probe_pkts_a2b, zwnd_probe_pkts_b2a,
                            zwnd_probe_bytes_a2b, zwnd_probe_bytes_b2a,
                            outoforder_pkts_a2b, outoforder_pkts_b2a,
                            pushed_data_pkts_a2b, pushed_data_pkts_b2a,
                            SYN_pkts_sent_a2b, FIN_pkts_sent_a2b,
                            SYN_pkts_sent_b2a, FIN_pkts_sent_b2a,
                            req_1323_ws_a2b, req_1323_ts_a2b,
                            req_1323_ws_b2a, req_1323_ts_b2a,
                            adv_wind_scale_a2b, adv_wind_scale_b2a,
                            req_sack_a2b, req_sack_b2a, sacks_sent_a2b,
                            sacks_sent_b2a, urgent_data_pkts_a2b,
                            urgent_data_pkts_b2a, urgent_data_bytes_a2b,
                            urgent_data_bytes_b2a, mss_requested_a2b,
```

```
                    mss_requested_b2a, max_segm_size_a2b,
                    max_segm_size_b2a, min_segm_size_a2b,
                    min_segm_size_b2a, avg_segm_size_a2b,
                    avg_segm_size_b2a, max_win_adv_a2b,
                    max_win_adv_b2a, min_win_adv_a2b,
                    min_win_adv_b2a, zero_win_adv_a2b,
                    zero_win_adv_b2a, avg_win_adv_a2b,
                    avg_win_adv_b2a, initial_window_bytes_a2b,
                    initial_window_bytes_b2a,
                    initial_window_pkts_a2b, initial_window_pkts_b2a
                    , ttl_stream_length_a2b, ttl_stream_length_b2a,
                    missed_data_a2b, missed_data_b2a,
                    truncated_data_a2b, truncated_data_b2a,
                    truncated_packets_a2b, truncated_packets_b2a,
                    data_xmit_time_a2b, data_xmit_time_b2a,
                    idletime_max_a2b, idletime_max_b2a,
                    hardware_dups_a2b, hardware_dups_b2a,
                    throughput_a2b, throughput_b2a, RTT_samples_a2b,
                     RTT_samples_b2a, RTT_min_a2b, RTT_min_b2a,
                    RTT_max_a2b, RTT_max_b2a, RTT_avg_a2b,
                    RTT_avg_b2a, RTT_stdev_a2b, RTT_stdev_b2a,
                    RTT_from_3WHS_a2b, RTT_from_3WHS_b2a,
                    RTT_full_sz_smpls_a2b, RTT_full_sz_smpls_b2a,
                    RTT_full_sz_min_a2b, RTT_full_sz_min_b2a,
                    RTT_full_sz_max_a2b, RTT_full_sz_max_b2a,
                    RTT_full_sz_avg_a2b, RTT_full_sz_avg_b2a,
                    RTT_full_sz_stdev_a2b, RTT_full_sz_stdev_b2a,
                    post_loss_acks_a2b, post_loss_acks_b2a,
                    ambiguous_acks_a2b, ambiguous_acks_b2a,
                    RTT_min_last_a2b, RTT_min_last_b2a,
                    RTT_max_last_a2b, RTT_max_last_b2a,
                    RTT_avg_last_a2b, RTT_avg_last_b2a,
                    RTT_sdv_last_a2b, RTT_sdv_last_b2a,
                    segs_cum_acked_a2b, segs_cum_acked_b2a,
                    duplicate_acks_a2b, duplicate_acks_b2a,
                    triple_dupacks_a2b, triple_dupacks_b2a,
                    max_retrans_a2b, max_retrans_b2a,
                    min_retr_time_a2b, min_retr_time_b2a,
                    max_retr_time_a2b, max_retr_time_b2a,
                    avg_retr_time_a2b, avg_retr_time_b2a,
                    sdv_retr_time_a2b, sdv_retr_time_b2a) VALUES (
                    $csdata)";
            }
            else{
                $sql_insert="INSERT INTO $tmp_table(host_a, host_b,
                    port_a, port_b, first_packet, last_packet,
                    total_packets_a2b, total_packets_b2a,
                    resets_sent_a2b, resets_sent_b2a,
                    ack_pkts_sent_a2b, ack_pkts_sent_b2a,
                    pure_acks_sent_a2b, pure_acks_sent_b2a,
                    sack_pkts_sent_a2b, sack_pkts_sent_b2a,
                    dsack_pkts_sent_a2b, dsack_pkts_sent_b2a,
                    max_sack_blks_ack_a2b, max_sack_blks_ack_b2a,
                    unique_bytes_sent_a2b, unique_bytes_sent_b2a,
                    actual_data_pkts_a2b, actual_data_pkts_b2a,
                    actual_data_bytes_a2b, actual_data_bytes_b2a,
```

```
                                        rexmt_data_pkts_a2b, rexmt_data_pkts_b2a,
                                        rexmt_data_bytes_a2b, rexmt_data_bytes_b2a,
                                        zwnd_probe_pkts_a2b, zwnd_probe_pkts_b2a,
                                        zwnd_probe_bytes_a2b, zwnd_probe_bytes_b2a,
                                        outoforder_pkts_a2b, outoforder_pkts_b2a,
                                        pushed_data_pkts_a2b, pushed_data_pkts_b2a,
                                        SYN_pkts_sent_a2b, FIN_pkts_sent_a2b,
                                        SYN_pkts_sent_b2a, FIN_pkts_sent_b2a,
                                        req_1323_ws_a2b, req_1323_ts_a2b,
                                        req_1323_ws_b2a, req_1323_ts_b2a,
                                        adv_wind_scale_a2b, adv_wind_scale_b2a,
                                        req_sack_a2b, req_sack_b2a, sacks_sent_a2b,
                                        sacks_sent_b2a, urgent_data_pkts_a2b,
                                        urgent_data_pkts_b2a, urgent_data_bytes_a2b,
                                        urgent_data_bytes_b2a, mss_requested_a2b,
                                        mss_requested_b2a, max_segm_size_a2b,
                                        max_segm_size_b2a, min_segm_size_a2b,
                                        min_segm_size_b2a, avg_segm_size_a2b,
                                        avg_segm_size_b2a, max_win_adv_a2b,
                                        max_win_adv_b2a, min_win_adv_a2b,
                                        min_win_adv_b2a, zero_win_adv_a2b,
                                        zero_win_adv_b2a, avg_win_adv_a2b,
                                        avg_win_adv_b2a, initial_window_bytes_a2b,
                                        initial_window_bytes_b2a,
                                        initial_window_pkts_a2b, initial_window_pkts_b2a
                                        , ttl_stream_length_a2b, ttl_stream_length_b2a,
                                        missed_data_a2b, missed_data_b2a,
                                        truncated_data_a2b, truncated_data_b2a,
                                        truncated_packets_a2b, truncated_packets_b2a,
                                        data_xmit_time_a2b, data_xmit_time_b2a,
                                        idletime_max_a2b, idletime_max_b2a,
                                        hardware_dups_a2b, hardware_dups_b2a,
                                        throughput_a2b, throughput_b2a) VALUES  ($csdata
                                        )";
                        }
                        my $sth=$dbh_g->prepare($sql_insert);
                        $sth->execute or die "SQL Error: $DBI::errstr\n";;
                }
        }
}

sub do_inter_packet_stats{
        my $filename=shift;
        my $tmp_table=shift;

        if (! -e $filename || $tmp_table eq ""){
                print "ERROR: do_inter_packet_stats being passed non existent file
                    or blank table name: \"$filename \".\n";
                exit;
        }

        my $log = Net::TcpDumpLog->new();
        $log->read($filename);

        my @Indexes = $log->indexes;
        my $index;
```

```
my ($length_orig,$length_incl,$drops,$secs,$msecs);
my $data;

my $flowDuration=0;
my @packetArrivalTimes=();
my @packetInterArrivalTimes=();
my @payloadSizes=();

#Data structures for storing session info to calc stats from
my %ip_pairs_packetArrivalTimes=();
my %ip_pairs_payloadSizes=();

#Stats vars
my $statArrivalTimes_a2b = Statistics::Descriptive::Full->new();
my $statArrivalTimes_b2a = Statistics::Descriptive::Full->new();
my $statPayloads_a2b = Statistics::Descriptive::Full->new();
my $statPayloads_b2a = Statistics::Descriptive::Full->new();

#IP Headers
my $ip_ver;
my $ip_hlen;    #header length
my $ip_flags;
my $ip_foffset;
my $ip_tos;
my $ip_len;     #length including header
my $ip_id;      #identification (seq) number for packet
my $ip_ttl;
my $ip_proto;   #ip protocol number
my $ip_cksum;
my $ip_src;
my $ip_dst;
my $ip_opt;
my $ip_data;    # Also defined above as $tcp_obj

#TCP Headers
my $tcp_srcPort;
my $tcp_dstPort;
my $tcp_seqNum;
my $tcp_acknum;
my $tcp_hlen;
my $tcp_rsvd;
my $tcp_flags;
my $tcp_winsize;
my $tcp_cksum;
my $tcp_urg;
my $tcp_options;
my $tcp_data;

my $numberOfPackets=@Indexes;

my ($ip_init,$ip_recv,$tcp_init,$tcp_recv);     #Who started the
    conversation?
my $first_packet_arrival=-1;

foreach $index (@Indexes) {
        ($length_orig,$length_incl,$drops,$secs,$msecs) = $log->header(
```

```
    $index);
#Pad zeroes in front of short msecs:
my $tmp=6-length($msecs);
for(my $i=1; $i<=$tmp; $i++){
        $msecs="0" . $msecs;
}

$data = $log->data($index);
my ($ether_dest,$ether_src,$ether_type,$ether_data) = unpack('
    H12H12H4a*',$data);

my $ip_obj   = NetPacket::IP->decode($ether_data);
my $tcp_obj  = NetPacket::TCP->decode( $ip_obj->{data} );

#Ethernet headers:
# length_orig, length_incl, $dtops, $secs, $msecs

#IP Headers
$ip_ver         = $ip_obj->{ver};
$ip_hlen        = $ip_obj->{hlen};      #header length
$ip_flags       = $ip_obj->{flags};
$ip_foffset     = $ip_obj->{foffset};
$ip_tos         = $ip_obj->{tos};
$ip_len         = $ip_obj->{len};       #length including header
$ip_id          = $ip_obj->{id};        #identification (seq) number
     for packet
$ip_ttl = $ip_obj->{ttl};
$ip_proto       = $ip_obj->{proto};     #ip protocol number
$ip_cksum       = $ip_obj->{cksum};
$ip_src         = $ip_obj->{src_ip};
$ip_dst         = $ip_obj->{dest_ip};
$ip_opt = $ip_obj->{options};
$ip_data        = $ip_obj->{data};      # Also defined above as
    $tcp_obj

#TCP Headers
$tcp_srcPort    = $tcp_obj->{src_port};
$tcp_dstPort    = $tcp_obj->{dest_port};
$tcp_seqNum     = $tcp_obj->{seqnum};
$tcp_acknum     = $tcp_obj->{acknum};
$tcp_hlen       = $tcp_obj->{hlen};
$tcp_rsvd       = $tcp_obj->{reserved};
$tcp_flags      = $tcp_obj->{flags};
$tcp_winsize    = $tcp_obj->{winsize};
$tcp_cksum      = $tcp_obj->{cksum};
$tcp_urg        = $tcp_obj->{urg};
$tcp_options    = $tcp_obj->{options};
$tcp_data       = $tcp_obj->{data};

#Set who started the conversation
if($first_packet_arrival == -1){
        $first_packet_arrival=$secs.".".$msecs;
        $ip_init=$ip_src;
        $ip_recv=$ip_dst;
        $tcp_init=$tcp_srcPort;
        $tcp_recv=$tcp_dstPort;
```

```
        }


        push @{ $ip_pairs_packetArrivalTimes{$ip_src . "." . $tcp_srcPort .
            "-" . $ip_dst . "." . $tcp_dstPort} }, $secs . "." .$msecs;
        push @{ $ip_pairs_payloadSizes{$ip_src . "-" . $ip_dst} }, $ip_len-
            $ip_hlen;

}

#Should only be two hosts in communication:
if( keys( %ip_pairs_packetArrivalTimes ) > 2){
        print "ERROR: Data file involves more than one TCP session. Have you
             demultiplexed?\n";
        print "Failed on $filename\n";

        print Dumper %ip_pairs_packetArrivalTimes;


        exit;
}

#Calcuate inter arrival times, ie the time difference between each packet's
    arrival
for my $ip_pair ( keys %ip_pairs_packetArrivalTimes ) {
        my $arr_ref = \@{ $ip_pairs_packetArrivalTimes{$ip_pair}};
        for(my $i=0; $i < @$arr_ref-1; $i++){
                @$arr_ref[$i] = @$arr_ref[$i+1]-@$arr_ref[$i] . " ";
        }
        pop @$arr_ref;
}

#Do statistics
my @kyz_time = keys(%ip_pairs_packetArrivalTimes);
my @kyz_payload =keys(%ip_pairs_payloadSizes);

my $T_mean_a2b = 0;
my $T_variance_a2b =0;
my $T_num_a2b =0;
my $T_stDv_a2b=0;
my $T_mean_b2a=0;
my $T_variance_b2a=0;
my $T_num_b2a=0;
my $T_stDv_b2a=0;
my $P_mean_a2b=0;
my $P_variance_a2b=0;
my $P_num_a2b=0;
my $P_stDv_a2b=0;
my $P_mean_b2a=0;
my $P_variance_b2a=0;
my $P_num_b2a=0;
my $P_stDv_b2a=0;


#TIME:
if(@{$ip_pairs_packetArrivalTimes{$kyz_time[0]}} > 1 ){
```

```
                                        #If we have more than 1 packet in the
            monologue of the session, otherwise =null
        $statArrivalTimes_a2b->add_data(\@{ $ip_pairs_packetArrivalTimes{$kyz_time
            [0]}});
                $T_mean_a2b     = $statArrivalTimes_a2b->mean();
                $T_variance_a2b  = $statArrivalTimes_a2b->variance();
                $T_num_a2b      = $statArrivalTimes_a2b->count();
                $T_stDv_a2b     = $statArrivalTimes_a2b->standard_deviation();
        }

#Bolted on test to see if two hosts are involved
# in communication- some traffic seems to only have one host
# which is very strange
if(keys( %ip_pairs_packetArrivalTimes )>1){
        if(@{$ip_pairs_packetArrivalTimes{$kyz_time[1]}} > 1){
        $statArrivalTimes_b2a->add_data(\@{ $ip_pairs_packetArrivalTimes{$kyz_time
            [1]}});
                $T_mean_b2a     = $statArrivalTimes_b2a->mean();
                $T_variance_b2a  = $statArrivalTimes_b2a->variance();
                $T_num_b2a      = $statArrivalTimes_b2a->count();
                $T_stDv_b2a     = $statArrivalTimes_b2a->standard_deviation();
        }
}

        #PAYLOAD:
        if(@{$ip_pairs_payloadSizes{$kyz_payload[0]}} > 1){
                $statPayloads_a2b->add_data(\@{ $ip_pairs_payloadSizes{$kyz_payload
                    [0]}});
                $P_mean_a2b     = $statPayloads_a2b->mean();
                $P_variance_a2b  = $statPayloads_a2b->variance();
                $P_num_a2b      = $statPayloads_a2b->count();
                $P_stDv_a2b     = $statPayloads_a2b->standard_deviation();
        }
if(keys( %ip_pairs_packetArrivalTimes )>1){
        if(@{$ip_pairs_payloadSizes{$kyz_payload[1]}} > 1){
        $statPayloads_b2a->add_data(\@{ $ip_pairs_payloadSizes{$kyz_payload[1]}});
                $P_mean_b2a     = $statPayloads_b2a->mean();
                $P_variance_b2a  = $statPayloads_b2a->variance();
                $P_num_b2a      = $statPayloads_b2a->count();
                $P_stDv_b2a     = $statPayloads_b2a->standard_deviation();
        }
}

        my $sql_iat = $dbh_g->prepare("UPDATE $tmp_table SET inter_packet_t_mean_a2b
            ='$T_mean_a2b',inter_packet_t_variance_a2b='$T_variance_a2b',
            inter_packet_t_stdev_a2b='$T_stDv_a2b',inter_packet_p_mean_a2b='
            $P_mean_a2b',inter_packet_p_variance_a2b='$P_variance_a2b',
            inter_packet_p_stdev_a2b='$P_stDv_a2b', inter_packet_t_mean_b2a='
            $T_mean_b2a',inter_packet_t_variance_b2a='$T_variance_b2a',
            inter_packet_t_stdev_b2a='$T_stDv_b2a',inter_packet_p_mean_b2a='
            $P_mean_b2a',inter_packet_p_variance_b2a='$P_variance_b2a',
            inter_packet_p_stdev_b2a='$P_stDv_b2a' WHERE host_a='$ip_init' AND
            host_b='$ip_recv' AND port_a='$tcp_init' AND port_b='$tcp_recv' AND
            first_packet>$first_packet_arrival-10 AND first_packet<
            $first_packet_arrival+10");
        $sql_iat->execute or die "SQL Error: $DBI::errstr\n";
```

```
}#end do_inter_packet_stats

sub tcpflow_it{

        #1. Extrace the payload of each .trace file. Each trace file will generate
            files of the form:
        #     163.001.236.180.47248-209.085.227.099.00080.trace
        #     srcIP.port-dstIP.port
        # only operate on .trace files, and ignore the one non_ip.trace one
        #print "#Looking at $File::Find::name\n";
        /\.trace$/ or return;                   #a. Only .trace files
        if(/non_ip.trace/){                     #b. Except non_ip.trace
                return;
        }
        if($File::Find::dir=~m/\/p(\d+)\//){    #c. And only TCP
                my $protocol_number=$1;
                if($protocol_number != 6){
                        #print "Not TCP traffic, ignoring ($protocol_number)\n";
                        $File::Find::prune=1;
                        return;
                }

        }
        else{
                print "ERROR: Cannot determine protocol number in use.. Exiting.\n";
                print "Offending file: $File::Find::name\n\n";
                exit;
        }

        if(++$display_counter % 50== 0){
                print "$display_counter payloads extracted\r";
        }

        my $filename=$_;
        my $result=`tcpflow -b $tcpflow_maxbytes -r $filename 2>>./tcpflow_stderr.
            txt`;


        #2. Get the ports and timestamp from the first packet in the trace file, and
             rename the file to be of the form
        my ($ip_src,$ip_dst,$tcp_srcPort,$tcp_dstPort,$timestamp)=("","","","","");
        if($filename=~m/(\d+\.\d+)-s(\d+)-d(\d+)/){
                ($timestamp,$tcp_srcPort,$tcp_dstPort) =($1,$2,$3);
        }
        else{
                print "ERROR: Bad .trace filename encountered when trying to extract
                     info:\n";
                print "\"$filename\"\n";
                exit;
        }

        #3. Get src and dest IPs
        if($File::Find::dir=~m/S(\d+\.\d+\.\d+\.\d+)\/D(\d+\.\d+\.\d+\.\d+)/){
                ($ip_src,$ip_dst)=($1,$2);
        }
```

```
        else{
                print "ERROR: Bad .trace directory structure encountered when trying
                     to extract info.\n";
                exit;
        }


        ($ip_src,$ip_dst,$tcp_srcPort,$tcp_dstPort) = &add_leading_zeroes($ip_src,
            $ip_dst,$tcp_srcPort,$tcp_dstPort);

        my $orig_filename_a2b="$ip_src.$tcp_srcPort-$ip_dst.$tcp_dstPort";
        my $orig_filename_b2a="$ip_dst.$tcp_dstPort-$ip_src.$tcp_srcPort";
        rename($orig_filename_a2b, "$timestamp-$orig_filename_a2b");
        rename($orig_filename_b2a, "$timestamp-$orig_filename_b2a");
}

sub build_list_of_darpa_tables{
        if($File::Find::dir=~m/(week\d)\/(monday|tuesday|wednesday|thursday|friday)
            \//){
                my $tbl_name="$tbl_prefix$1_$2$tbl_postfix";
                #$table_list{$tbl_name}=1;

        }
}

### Drop table if exists, then creates
sub create_tcptrace_table{

# Not included for brevity- this subroutine creates the table described in Appendix
    A, above.

}

sub file_content_inspector{
    my $filename=shift;


    my $file = read_file($filename, binmode => ':raw' ) ;
    my %bigrams=();
    my %unigrams=();
    my $total=0;
    my $H_uni=0;        #Entropy
    my $H_bi=0;

    $total=length $file;

    #Content features
    my $most_freq_char=0;
    my $count_of_mfc=0;
    my $num_of_zeroes=0;                                # Too many zeroes may be an
        indiator of a NOP slide?
    my $proportion_zeroes=0;
    my $num_of_ascii_printable=0;
    my $num_of_ascii_non_printable=0;
    my $prop_printable=0;
    my $prop_non_printable=0;
```

```
#New
my $proportion_mfc=0;
my $most_freq_bigram=0;
my $count_of_mf_bi=0;
my $proportion_mf_bigram=0;

#Ignore empty files
if($total>0){

    #Build unigrams/bigrams
    for(my $i=0; $i<$total-1; $i++){
        my $bigram = substr $file, $i, 2;
        my $unigram = substr $file, $i,1;
        $bigrams{$bigram}++;
        $unigrams{$unigram}++;
    }
    $unigrams{substr $file, $total-1,1}++;

    #1. Calculate unigram entropy
    foreach my $unigram(keys %unigrams){
        #a. Entropy
        my $p=$unigrams{$unigram}/$total;
        $H_uni+=$p*log($p);

        #b. Most freq char
        if($unigrams{$unigram} > $count_of_mfc){
                $most_freq_char=ord($unigram);
                $count_of_mfc=$unigrams{$unigram};
        }
        #c. printable chars
        if(ord($unigram)>=32 && ord($unigram)<=126){
            $num_of_ascii_printable+=$unigrams{$unigram};
        }
        else{
            $num_of_ascii_non_printable+=$unigrams{$unigram};
        }
    }
    $proportion_mfc=$count_of_mfc / $total;
    $H_uni=-$H_uni/log(2);

    #c.2 Proportion of printable/non chars
    if($total>0){    #Emtpy file could create div by 0 ?
        $prop_printable=$num_of_ascii_printable/$total;
        $prop_non_printable=$num_of_ascii_non_printable/$total;
    }

    #d. Number of zeroes
    if(exists($unigrams{'0'})){
            $num_of_zeroes=$unigrams{'0'};
            $proportion_zeroes=$num_of_zeroes / $total;
    }
    else{
            $num_of_zeroes=0;
            $proportion_zeroes=0;
    }
```

```
        #Calculate bigram entropy
        foreach my $bigram(keys %bigrams){
                #1. Entropy
                my $p=$bigrams{$bigram}/$total;
                $H_bi+=$p*log($p);
                #b. Most freq bigram:
                if($bigrams{$bigram} > $count_of_mf_bi){
                        $most_freq_bigram=sprintf("%03d",ord(substr ($bigram, 0,1)))
                             . sprintf("%03d",ord(substr($bigram, 1,1)));   #e.g 'he
                            ' =>104 . 101 (zero padding to)
                        #$most_freq_bigram=$bigram;
                        $count_of_mf_bi=$bigrams{$bigram};
                }
        }
        $H_bi=-$H_bi/log(2);
        $proportion_mf_bigram=$count_of_mf_bi / $total;

    }
    else{
        print "WARNING: File has no contents - \"$filename\"\n";
    }
    return ($total,$H_uni,$most_freq_char,$count_of_mfc,$proportion_mfc,
        $num_of_zeroes,$proportion_zeroes,$num_of_ascii_printable,
        $num_of_ascii_non_printable,$prop_printable,$prop_non_printable,$H_bi,
        $most_freq_bigram,$count_of_mf_bi,$proportion_mf_bigram);
}

#Pass table,column,type
sub check_table_existence{
        my $tmpTable=shift;
        my $sth_tb=$dbh_g->prepare("SHOW TABLES LIKE '$tmpTable'");
        $sth_tb->execute or die "SQL Error: $DBI::errstr\n";
        if($sth_tb->fetchrow() eq ""){
                return 0;
        }
        else{
                return 1;
        }
        return -1;
}


#Convert tcpflow's filenames of 163.001.236.180.47248-209.085.227.099.00080 to
    163.1.236.180.47248-209.85.227.99.80
sub remove_leading_zeroes{
        my $filename=shift;

                my @points=split('-',$filename);
                my $src=$points[0];
                my $dst=$points[1];


                my @tmp=split(/\./,$src);
                my $ip_src="$tmp[0].$tmp[1].$tmp[2].$tmp[3]";
                my $tcp_srcPort=$tmp[4];
```

```
                @tmp=split(/\./,$dst);
                my $ip_dst="$tmp[0].$tmp[1].$tmp[2].$tmp[3]";
                my $tcp_dstPort=$tmp[4];

                $ip_src=~m/(...)\.(...)\.(...)\.(...)/;
                my($w,$x,$y,$z)=($1,$2,$3,$4);
                $w=~s/^0*(\d)/$1/; $x=~s/^0*(\d)/$1/; $y=~s/^0*(\d)/$1/; $z=~s/^0*(\
                    d)/$1/;
                $ip_src="$w.$x.$y.$z";
                $tcp_srcPort=~s/^0*(\d)/$1/;

                $ip_dst=~m/(...)\.(...)\.(...)\.(...)/;
                ($w,$x,$y,$z)=($1,$2,$3,$4);
                $w=~s/^0*(\d)/$1/; $x=~s/^0*(\d)/$1/; $y=~s/^0*(\d)/$1/; $z=~s/^0*(\
                    d)/$1/;
                $ip_dst="$w.$x.$y.$z";
                $tcp_dstPort=~s/^0*(\d)/$1/;

                return ($ip_src, $ip_dst, $tcp_srcPort, $tcp_dstPort);
}

sub add_leading_zeroes{
        my ($ip_src,$ip_dst,$tcp_srcPort,$tcp_dstPort)=@_;

        my @ip=split(/\./,$ip_src);
        foreach my $octet (@ip){
                if(length($octet) == 1){ $octet="00" . $octet;}
                if(length($octet) == 2){ $octet="0" . $octet;}
        }
        $ip_src=join('.',@ip);
        @ip=split(/\./,$ip_dst);
        foreach my $octet (@ip){
                if(length($octet) == 1){ $octet="00" . $octet;}
                if(length($octet) == 2){ $octet="0" . $octet;}
        }
        $ip_dst=join('.',@ip);

        my $tmp=5-length($tcp_srcPort);
        for(my $i=1; $i<=$tmp; $i++){
                $tcp_srcPort="0" . $tcp_srcPort;
        }

        $tmp=5-length($tcp_dstPort);
        for(my $i=1; $i<=$tmp; $i++){
                $tcp_dstPort="0" . $tcp_dstPort;
        }

        #print "Y > $ip_src,$ip_dst,$tcp_srcPort,$tcp_dstPort\n\n";
        return($ip_src,$ip_dst,$tcp_srcPort,$tcp_dstPort);

}
```

C.2 hostileTrafficImporter.pl

```
#!/usr/bin/perl -w
use strict;
use File::Find;
use DBI;
use Time::Local;

my $dir="";
my $join_tables=0;
my $darpa=0;
my ($tbl_prefix,$tbl_postfix)=("","");
my $as_filename=0;
my %table_list=();
my $prefix="";

#DB variables
my $database="DBI:mysql:project";
my $username="user";
my $password="12d2ded7bba8008cc176cc6e7ff2d619";
my $dbh_g;

if($#ARGV<1){
        usage();
        exit;
}

#Parse args
for(my $i=0; $i<$#ARGV+1;$i++){
        if($ARGV[$i] eq "-darpa"){
                $darpa=1;
        }
        elsif($ARGV[$i] eq "-dir"){
                $dir=$ARGV[$i+1];$i++;
                if(! -e $dir){
                        print "ERROR: $dir does not exist\n";
                        exit;
                }
        }
        elsif($ARGV[$i] eq "-as_filename"){
                $as_filename=1;
        }
        elsif($ARGV[$i] eq "-tbl_prefix"){
                $tbl_prefix=$ARGV[$i+1];$i++;
        }
        elsif($ARGV[$i] eq "-tbl_postfix"){
                $tbl_postfix=$ARGV[$i+1];$i++;
        }
        elsif($ARGV[$i] eq "-join_tables"){
                $join_tables=1;
        }
        elsif($ARGV[$i] eq "-join_prefix"){
                $prefix=$ARGV[$i+1];$i++;
        }
        else{
                print "ERROR: Unknown arg - $ARGV[$i]\n";
```

```
                exit;
        }
}


if($join_tables==0 && ($as_filename == 0 && $darpa == 0) || ($as_filename == 1 &&
    $darpa == 1)){
        print "ERROR: Use one of -darpa or -as_filename for output\n";
        exit;
}

$dbh_g = DBI->connect($database, $username, $password) || die "Could not connect to
    database: $DBI::errstr";
my @trace_table;
my @attack_table;

if($join_tables==0){
        print "##Building attack list tables from tcpout.list files\n";
        my $display_counter=0;
        find(\&find_attack_list_files,$dir);
        print "\n# Done\n";
}
else{
        print "Joining tables based on $prefix prefix.\n";
        #Build a list of all tables matching the prefix criteria
        my $sth_gt=$dbh_g->prepare("SHOW TABLES");
        $sth_gt->execute() or die "SQL Error: $DBI::errstr\n";
        while(my $row = $sth_gt->fetchrow()){
                $table_list{$row}=1;
                my $tmp=$prefix . "_attackList";
                #print "matching $row\n";
                if($row =~m/$prefix$/){
                        push(@trace_table,$row);
                        #print "reg tab - $row\n";
                }
                elsif($row =~m/$tmp/){
                        push(@attack_table,$row);
                        #print "attach tab - $row\n";
                }
                else{
                        print "Ignoring table \"$row\"\n";
                        #exit;
                }
        }
        if(@trace_table != @attack_table){
                print "ERROR: I didn't manage to match up all tables\n";
        }

        for(my $i=0; $i<@trace_table;$i++){
                &append_attack_name($trace_table[$i],$trace_table[$i]."_attackList")
                    ;
        }
}

$dbh_g->disconnect();
```

```
sub usage{
print "This script looks for tcpdump.list or tcpout.list files from the DARPA 1998
    training set.
It parses each file and imports the info to a table. It also merges these tables
    with the output
from tcptrace tables (created by flow_parser.pl)\n\n";
        print "Usage:\n";
        print "(populate DB) perl hostileTrafficImporter.pl -dir <dir_to_search> [-
            darpa |-as_filename] -tbl_prefix <prefix> -tbl_postfix <postfix>\n";
        print "(join tables) perl hostileTrafficImporter.pl -join_tables -
            join_prefix <regular_expression>\n\n";
        print "e.g perl hostileTrafficImporter_new.pl -dir ./DARPA_data/ -darpa -
            tbl_prefix 1998\n";
        print "e.g perl hostileTrafficImporter_new.pl -join_tables -join_prefix \"
            week\\d_(wednesday|monday|tuesday)\"\n\n";
}

#Attack list files should end in .list
sub find_attack_list_files{
        #Ignore millions of traced files in demux folders:
        my $pathname=$File::Find::name;
        my $filename=$_;
        if($File::Find::dir=~m/demux/){
                $File::Find::prune=1;
                return;
        }
        /(tcpout|tcpdump)\.(list|lllist)/ or return;
        my $tmp_table=&get_attack_tablename($pathname,$filename);
        print "$pathname - $tmp_table\n";
        &populate_attack_list($filename,$tmp_table);
}

sub get_attack_tablename{

        my $pathname=shift;
        my $filename=shift;
        my $tmpTableName="";

        if($darpa == 1){
                if($pathname=~m/(week\d)\/(monday|tuesday|wednesday|thursday|friday)
                    \//){
                        $tmpTableName=$tbl_prefix . $1 ."_". $2 . "_attackList" .
                            $tbl_postfix;
                }
                else{
                        print "##ERROR: DARPA folders not in required format of week
                            <n>/<day> - \"$pathname\"\n";
                        exit;
                }
        }
        elsif($as_filename==1){
                if($filename=~/(.*?)\.list/){
                        $tmpTableName=$tbl_prefix . $1 . "_attackList" .
                            $tbl_postfix;
                }
                else{
```

```
                        print "ERROR: Cannot get filename\n"; #Should be impossible
                            to reach here
                        exit;
                }
        }
}

#Creates an attack list table from 1998 DARPA provided tcpout.list files
sub populate_attack_list{
        my $inputFile=shift;
        my $tmp_table=shift;

        my $epochTime;
        my ($month,$day,$year);
        my ($hrs,$mins,$secs);
        my ($duration,$service);
        my ($tcp_srcPort,$tcp_dstPort,$ip_src,$ip_dst);
        my ($attack_score,$attack_name);
        my $flag=-1;

        open(FILE_IN,"<".$inputFile) or die "Error opening \"$inputFile\"\n";

        #Create a table for the data:
        my $sth_ct_al=$dbh_g->prepare("drop table if exists $tmp_table");
        $sth_ct_al->execute or die "SQL Error: $DBI::errstr\n";
        $sth_ct_al=$dbh_g->prepare("CREATE TABLE $tmp_table (date VARCHAR(20),
            duration VARCHAR(20), service VARCHAR(10), tcp_srcPort INT, tcp_dstPort
            INT,ip_src VARCHAR(15),ip_dst VARCHAR(15),attack_score INT,attack_name
            VARCHAR(100))");
        $sth_ct_al->execute or die "SQL Error: $DBI::errstr\n";

        #Read the attack list
        while(my $record=<FILE_IN>){
                chomp($record);
                my @fields = split(" ", $record);
                if($fields[1]=~m/(..)\/(..)\/(....)/){          #01/23/1998
                        $month=$1-1; $day=$2; $year=$3;
                }
                elsif($fields[1]=~m/(..)\/(..)\/(..)/){         #01/23/98
                        $month=$1-1; $day=$2; $year="19".$3;
                }
                else{
                        print "Date seems imparsable\n";
                        exit;
                }
                $fields[2]=~m/(..):(..):(..)/;                  #16:56:12
                $hrs=$1; $mins=$2; $secs=$3;
                $duration=$fields[3];                           #00:01:26
                $service=$fields[4];                            #http or dns/u ::
                    Should check
                $tcp_srcPort=$fields[5];
                $tcp_dstPort=$fields[6];
                $ip_src=$fields[7];
                $ip_dst=$fields[8];
                $attack_score=$fields[9];
                $attack_name=$fields[10];
```

```
                ##NUANCES:
                #########

                #1. IP Addresses have annoying 0's inside them.
                $ip_src=~m/(.+)\.(.+)\.(.+)\.(.+)/;
                my($w,$x,$y,$z)=($1,$2,$3,$4);
                $w=~s/^0*(\d)/$1/; $x=~s/^0*(\d)/$1/; $y=~s/^0*(\d)/$1/; $z=~s/^0*(\
                    d)/$1/;
                $ip_src="$w.$x.$y.$z";

                $ip_dst=~m/(.+)\.(.+)\.(.+)\.(.+)/;
                ($w,$x,$y,$z)=($1,$2,$3,$4);
                $w=~s/^0*(\d)/$1/; $x=~s/^0*(\d)/$1/; $y=~s/^0*(\d)/$1/; $z=~s/^0*(\
                    d)/$1/;
                $ip_dst="$w.$x.$y.$z";

                # 1. There's an error in the packet listing, the 32nd of July should
                     be the 1st August
                if($day eq "32" && $month eq "6" && $year eq "1998"){
                        $month="07";
                        $day="1";
                }


                #2. The pcap file was captured in a different time zone, and we
                    therefore get a discepency
                #       of 5 hours between the pcap files and the tcpdump.list files
                    .

                $epochTime = timelocal($secs, $mins,$hrs,$day,$month,$year) + 18000;
                     #21600=6 hours; #18000 seconds = 5 hours

                my $sth_pop_al=$dbh_g->prepare("INSERT INTO $tmp_table (date,
                    duration,service,tcp_srcPort,tcp_dstPort,ip_src,ip_dst,
                    attack_score,attack_name) VALUES ('$epochTime', '$duration', '
                    $service', '$tcp_srcPort', '$tcp_dstPort', '$ip_src', '$ip_dst',
                     '$attack_score', '$attack_name')");
                $sth_pop_al->execute or die "SQL Error: $DBI::errstr\n";

        }

        close(FILE_IN);

}


sub append_attack_name{
    #Match within 1.5 mins either way
    #We need the LIMIT 1 as the data is dirty in places, with the same connection
        being reported several times
    # within a few seconds (i.e identical ip_src,ip_dst,port_src,port_dst with small
         delta in first packet timestamp)
    #1. Check how many TCP attacks are listed
    my $table=shift;
    my $table_attack_list=shift;
```

```
    my $sth_attack_stats=$dbh_g->prepare("SELECT COUNT(*) FROM $table_attack_list
        WHERE service LIKE '%/%'");
    $sth_attack_stats->execute();
    my $count_other_attacks=$sth_attack_stats->fetchrow();
    $sth_attack_stats=$dbh_g->prepare("SELECT COUNT(*) FROM $table_attack_list WHERE
         service NOT LIKE '%/%'");
    $sth_attack_stats->execute();
    my $count_tcp_attacks=$sth_attack_stats->fetchrow();

    my $sth_total_tcptrace = $dbh_g->prepare("SELECT COUNT(*) FROM $table");
    $sth_total_tcptrace->execute();
    my $count_all_trace=$sth_total_tcptrace->fetchrow();

    print "Table $table_attack_list: $count_tcp_attacks TCP attacks (
        $count_other_attacks ICMP,UDP - ignored)\n";
    print "Table $table: $count_all_trace sessions\n";
    print "Updating tcptrace table with attack names\n";
   # print "\n\nUPDATE $table SET attack_name=(SELECT attack_name FROM
       $table_attack_list WHERE host_a=ip_src AND host_b=ip_dst AND port_a=
       tcp_srcPort AND port_b=tcp_dstPort AND TRUNCATE(date,0) > first_packet-150
       AND TRUNCATE(date,0) < first_packet+150 LIMIT 1)\n\n";
    my $sth_attack_name=$dbh_g->prepare("UPDATE $table SET attack_name=(SELECT
        attack_name FROM $table_attack_list WHERE host_a=ip_src AND host_b=ip_dst
        AND port_a=tcp_srcPort AND port_b=tcp_dstPort AND TRUNCATE(date,0) >
        first_packet-150 AND TRUNCATE(date,0) < first_packet+150 LIMIT 1),
        attack_score=(SELECT attack_score FROM $table_attack_list WHERE host_a=
        ip_src AND host_b=ip_dst AND port_a=tcp_srcPort AND port_b=tcp_dstPort AND
        TRUNCATE(date,0) > first_packet-150 AND TRUNCATE(date,0) < first_packet+150
        LIMIT 1)");
    $sth_attack_name->execute();

    #Let's check how many matches there were
    $sth_attack_stats=$dbh_g->prepare("SELECT COUNT(*) FROM $table WHERE attack_name
         IS NOT NULL");
    $sth_attack_stats->execute();
    my $count_outcome=$sth_attack_stats->fetchrow();
    print "$count_outcome rows out of $count_all_trace successfully tagged\n\n";
}
```

C.3 calculate_interflow.pl

```
#!/usr/bin/perl -w
use strict;
use DBI;

#######
# Adds columns to tcptrace table. The new data analyzes surrounding rows. So far
    does:
# total_conns_10s_anyport           #N.B The 10s interval can be set via the
    $interval variable
# total_conns_10s_sameServerPort
# total_packets_a2b_10s
# total_packets_b2a_10s
#
# Takes table name either via command line, or via a file list of table names.
#######

select(STDERR); $| = 1;          # make unbuffered
select(STDOUT); $| = 1;          # make unbuffered

##DB vars
my $database="DBI:mysql:project4";
my $username="user";
my $password="12d2ded7bba8008cc176cc6e7ff2d619";

#my $table="tbl_sd_tcptrace";
#my $table="";
my $interval=10;            #10 second range
my $VERBOSE=1;
my $prefix="";


print "This script manipulates tcptrace tables generated by flow_parser.pl program\n
    \n";
##Check command line options
if ($#ARGV+1 <1){
        print "ERROR: Not enough arguments\n";
        &usage;
        exit;
}

for(my $i=0; $i<$#ARGV+1;$i++){
    if($ARGV[$i] eq "-prefix"){
        $prefix=$ARGV[$i+1];$i++;
    }
    else{
        print "ERROR: Unknown argument \"$ARGV[$i]\". Exiting.\n";
        exit;
    }
}


##Connect to the db:
my $dbh_g = DBI->connect($database, $username, $password) || die "Could not connect
    to database: $DBI::errstr";
```

```
###Do stuff

my $sth_gt=$dbh_g->prepare("SHOW TABLES");
$sth_gt->execute() or die "SQL Error: $DBI::errstr\n";
while(my $row=$sth_gt->fetchrow()){

    if($row =~m/$prefix$/){
        print "Calculating total_conns_10s_anyport, total_conns_10s_sameServerPort,
            total_packets_a2b/b2a_10m for table \"$row\":\n";
        #&do_total_conns_btwn_hosts_10seconds_anyport($row);
        #&do_total_conns_btwn_hosts_10seconds_sameserverport($row);
        &do_total_bytes_btwn_hosts_10seconds($row);
    }
}


##Disconnect from the db
$dbh_g->disconnect();

sub usage{
    print "Usage:\n";
    print "e.g perl manip_tcptrace_tables.pl -prefix \"week\\d_(monday|tuesday|
        wednesday)\"\n\n";

}


sub do_total_conns_btwn_hosts_10seconds_anyport{
    my $table=shift;

    my $sth_001 = $dbh_g->prepare("SELECT first_packet,host_a,host_b,port_a,port_b
        FROM $table WHERE first_packet IS NOT NULL ORDER BY first_packet");
    $sth_001->execute or die "SQL Error: $DBI::errstr\n";
    my $num_of_conns;
    my @count_result;
    my $x=0;            #Used for verbose output- counts the number of insertions
        happening
    my $total_rows;

    ##Let's work out how many rows we have to do, phew!
    my $sth_total_rows=$dbh_g->prepare("SELECT COUNT(*) FROM $table");
    $sth_total_rows->execute();
    $total_rows = $sth_total_rows->fetchrow();

#    print "Total rows in \"$table\": $total_rows. Progress:\n";
#    print "Calculating total_conns_10s_anyport for table \"$table\":\n";
    while (my @row = $sth_001->fetchrow_array()) {  #Go through each row of the
        table
        my($fpx,$host_ax,$host_bx,$port_ax,$port_bx)= @row;
        #Create a new query to work out what machines this row has spoken to in the
            last 10 seconds:
        my $sth_002=$dbh_g->prepare("SELECT COUNT(*) FROM $table WHERE host_a='
            $host_ax' AND host_b='$host_bx' AND first_packet IS NOT NULL AND
            first_packet>$fpx-$interval AND first_packet<$fpx");
        $sth_002->execute or die "SQL Error: $DBI::errstr\n";
```

```
        while(@count_result = $sth_002->fetchrow_array()){
            $num_of_conns= $count_result[0];
        }

    #Third query to enter the value into the relevant row
    my $sth_003 = $dbh_g->prepare("UPDATE $table SET total_conns_$interval" . "
        s_anyport  = '$num_of_conns' WHERE host_a='$host_ax' AND host_b='$host_bx'
        AND port_a=$port_ax AND port_b=$port_bx AND first_packet=$fpx");
    $sth_003->execute or die "SQL Error: $DBI::errstr\n";

    if($VERBOSE){
        $x++;
        if(($x % 100)==0 || $x == $total_rows){
            print "$x/$total_rows\r";
        }
    }

    }
    print "\n";
}

sub do_total_conns_btwn_hosts_10seconds_sameserverport{
    my $table=shift;

    my $sth_001 = $dbh_g->prepare("SELECT first_packet,host_a,host_b,port_a,port_b
        FROM $table WHERE first_packet IS NOT NULL ORDER BY first_packet");
    $sth_001->execute or die "SQL Error: $DBI::errstr\n";
    my $num_of_conns;
    my @count_result;
    my $x=0;            #Used for verbose output- counts the number of insertions
        happening
    my $total_rows;

    ##Let's work out how many rows we have to do, phew!
    my $sth_total_rows=$dbh_g->prepare("SELECT COUNT(*) FROM $table");
    $sth_total_rows->execute();
    $total_rows = $sth_total_rows->fetchrow();

    #print "Total rows in \"$table\": $total_rows. Progress:\n";
    #print "Calculating total_conns_10s_sameServerPort for table \"$table\":\n";
    while (my @row = $sth_001->fetchrow_array()) {  #Go through each row of the
        table
        my($fpx,$host_ax,$host_bx,$port_ax,$port_bx)= @row;
        #Create a new query to work out what machines this row has spoken to in the
            last 10 seconds:
        my $sth_002=$dbh_g->prepare("SELECT COUNT(*) FROM $table WHERE host_a='
            $host_ax' AND host_b='$host_bx' AND port_b=$port_bx AND first_packet IS
            NOT NULL AND first_packet>$fpx-$interval AND first_packet<$fpx");
        $sth_002->execute or die "SQL Error: $DBI::errstr\n";
        while(@count_result = $sth_002->fetchrow_array()){
            $num_of_conns= $count_result[0];
        }

    #Third query to enter the value into the relevant row
    my $sth_003 = $dbh_g->prepare("UPDATE $table SET total_conns_$interval" ."
        s_sameServerPort  = '$num_of_conns' WHERE host_a='$host_ax' AND host_b='
```

```
        $host_bx' AND port_a=$port_ax AND port_b=$port_bx AND first_packet=$fpx");
    $sth_003->execute or die "SQL Error: $DBI::errstr\n";

    if($VERBOSE){
        $x++;
        if(($x % 100)==0 || $x == $total_rows){
            print "$x/$total_rows\r";
        }
    }

    }
    print "\n";
}

sub do_total_bytes_btwn_hosts_10seconds{
    my $table=shift;

    my $sth_001 = $dbh_g->prepare("SELECT first_packet,host_a,host_b,port_a,port_b,
        total_packets_a2b,total_packets_b2a FROM $table WHERE first_packet IS NOT
        NULL ORDER BY first_packet");
    $sth_001->execute or die "SQL Error: $DBI::errstr\n";
    my $num_of_conns;
    my @count_result;
    my $x=0;            #Used for verbose output- counts the number of insertions
        happening
    my $total_rows;

    ##Let's work out how many rows we have to do, phew!
    my $sth_total_rows=$dbh_g->prepare("SELECT COUNT(*) FROM $table");
    $sth_total_rows->execute();
    $total_rows = $sth_total_rows->fetchrow();

    #print "Total rows in $table: $total_rows. Progress:\n";
    #print "Calculating total_packets_a2b/b2a for table \"$table\":\n";
    while (my @row = $sth_001->fetchrow_array()) {  #Go through each row of the
        table
        my($fpx,$host_ax,$host_bx,$port_ax,$port_bx,$total_packets_a2bx,
            $total_packets_b2ax)= @row;
        my $sth_002=$dbh_g->prepare("SELECT total_packets_a2b,total_packets_b2a FROM
             $table WHERE host_a='$host_ax' AND host_b='$host_bx' AND first_packet
            IS NOT NULL AND first_packet>$fpx-10 AND first_packet<=$fpx");
        $sth_002->execute or die "SQL Error: $DBI::errstr\n";

        my $a2b_count=0;
        my $b2a_count=0;
        while(my @inner_row = $sth_002->fetchrow_array()){
            $a2b_count= $a2b_count + $inner_row[0];
            $b2a_count= $b2a_count + $inner_row[1];
        }

        my $update1="UPDATE $table SET total_packets_a2b_$interval" . "s = '
            $a2b_count' WHERE host_a='$host_ax' AND host_b='$host_bx' AND port_a=
            $port_ax AND port_b=$port_bx AND first_packet=$fpx";
        my $update2="UPDATE $table SET total_packets_b2a_$interval" . "s = '
            $b2a_count' WHERE host_a='$host_ax' AND host_b='$host_bx' AND port_a=
            $port_ax AND port_b=$port_bx AND first_packet=$fpx";
```

```
        my $sth_003 = $dbh_g->prepare($update1);
        $sth_003->execute or die "SQL Error: $DBI::errstr\n";
        $sth_003 = $dbh_g->prepare($update2);
        $sth_003->execute or die "SQL Error: $DBI::errstr\n";

        if($VERBOSE){
            $x++;
            if(($x % 100)==0 || $x == $total_rows){
                print "$x/$total_rows\r";
            }
        }
    }
    print "\n";
}
```

C.4 export_to_weka.pl

```
#!/usr/bin/perl
use strict;
use DBI;


#DB variables
my $database="DBI:mysql:project";
my $username="user";
my $password="12d2ded7bba8008cc176cc6e7ff2d619";
my $dbh_g;              #global db handle

my $prefix="";
my @c_names=();
my @c_types=();
my $ip_to_chars=1;          #Convert IP addresses into a,b,..,z,aa,ab,..,..,zzzzzz
my @ip_seqs="a".."zzzz";
my $ip_seqs_count=0;
my %ip_seqs_hash=();

#Count attacks
my %attacks=();
my $count_attacks=0;
my $count_non_attacks=0;

if($#ARGV+1 <1){
    print "Usage: perl export_to_arff.pl -prefix <table prefix>\n\n";
    exit;
}

#Parse args
for(my $i=0; $i<$#ARGV+1;$i++){
        if($ARGV[$i] eq "-prefix"){
                $prefix=$ARGV[$i+1];$i++;
        }
        elsif($ARGV[$i] eq "-"){
        }
        else{
            print "ERROR: Unknown arg - $ARGV[$i]\n";
            exit;
        }
}

#open db
$dbh_g = DBI->connect($database, $username, $password) || die "Could not connect to
    database: $DBI::errstr";

##1. Make sure all tables are identical
&check_tables_for_identicalness();

##2.

sub by_score { $attacks{$b} <=> $attacks{$a} }
my @sorted=sort by_score keys %attacks;
print "I observed the following from the tables:\n";
```

```
print "=>Total sessions: ". ($count_attacks+$count_non_attacks) . "\n";
print "=>Total attacks: $count_attacks\n";
print "=>Total non: $count_non_attacks\n";
print "=>Total types of attack: " . keys(%attacks) . "\n";

foreach my $attack (@sorted){
    print " ->$attack - $attacks{$attack}\n";
}

print "\n";

##3
my $response="";
while($response ne "n" && $response ne "e"){
    print "Do you want to select create a [n]ew config file, or read from an [e]
        xisting config file? [n/e]";
    $response=<STDIN>;
    chomp($response);
}

if($response eq "n"){
    &manual_selection();
}
elsif($response eq "e"){
    my $configfilename="";
    while($configfilename eq ""){
        print "Enter config file name please: ";
        $configfilename=<STDIN>;
        chomp($configfilename);
        if(!-e $configfilename){
            print "\"$configfilename\" does not exist\n\n";
            $configfilename="";
        }
    }
    print "Reading $configfilename\n";
    open(FILE,$configfilename);
    while (my $line = <FILE>){
        my @x=split(/,/,$line);
        push(@c_names,$x[0]);
        push(@c_types,$x[1]);
    }
    close FILE;
}
my $list_of_names=join(',',@c_names);

#Let's check for host_a or host_b. Used to convert IPs to short char seqs
my @ip_positions;
for(my $i=0; $i<@c_names; $i++){
    if($c_names[$i]=~m/host_(a|b)/){
        push(@ip_positions,$i);
    }
}

#4. Now c_names and c_types is populated.
print "Creating .arf files:\n";
my $sth_gt=$dbh_g->prepare("SHOW TABLES");
```

```
$sth_gt->execute() or die "SQL Error: $DBI::errstr\n";
while(my $row=$sth_gt->fetchrow()){

    if($row =~m/$prefix$/){
        print "=>Writing $row.arff\n";
        open(FILE,">$row.arff") or die "Cannot open $row\n";
        print FILE "\@relation $row\n\n";
        for(my $i=0; $i<@c_names; $i++){
            print FILE "\@attribute $c_names[$i]\t$c_types[$i]";
        }
        print FILE "\n\@data\n";
        my $sth_gd = $dbh_g->prepare("SELECT $list_of_names FROM $row");
        $sth_gd->execute or die "SQL Error: $DBI::errstr\n";
        while(my @row_data=$sth_gd->fetchrow_array()){
            #Change IP addresses into short letter references
            if($ip_to_chars ==1){
                foreach(@ip_positions){
                    if(exists $ip_seqs_hash{$row_data[$_]}){
                        $row_data[$_]=$ip_seqs_hash{$row_data[$_]};
                    }
                    else{
                        $ip_seqs_hash{$row_data[$_]}=$ip_seqs[$ip_seqs_count++];
                        $row_data[$_]=$ip_seqs_hash{$row_data[$_]};
                        if($ip_seqs_count>=456976){
                            print "ERROR: I've run out of IP addresses character
                                sequence codes (456,976 was my limit)\n";
                            exit;
                        }
                    }
                }
            }

            my $line=join(',',@row_data);
            print FILE "$line\n";
    #         for (my $i=0; $i<@row_data; $i++){
    #            print FILE "$row_data[$i]($c_names[$i]) ";
    #         }
    #         print FILE "\n\n\n";
        }

        close(FILE);

    }
    else{
        #print "Ignoring $row\n";
    }

}

if($ip_to_chars==1){
    print "Writing IP=>Char_seq mapping to ip_to_char.txt\n";
    open(FILE,">ip_to_char.txt");
    while ( my ($key, $value) = each(%ip_seqs_hash) ) {
        print FILE "$key,$value\n";
    }
}
```

```
#end db
$dbh_g->disconnect();


sub count_attacks{
    my $table=shift;

    my $sth_ca=$dbh_g->prepare("SELECT attack_name FROM $table WHERE attack_name IS
        NOT NULL");
    $sth_ca->execute() or die "\nSQL Error: $DBI::errstr\n";
    while(my $attack = $sth_ca->fetchrow()){
        if($attack ne "-"){
            $attacks{$attack}++;
            $count_attacks++;
        }
        else{
            $count_non_attacks++;
        }
    }

    #sub by_score { $attacks{$a} <=> $attacks{$b} }
    #my @sorted=sort by_score keys %attacks;
}
sub manual_selection{

    my $outputfile="";
    my $response="";
    my $sth_gt=$dbh_g->prepare("SHOW TABLES");
    $sth_gt->execute() or die "SQL Error: $DBI::errstr\n";

    my $row_to_use="";
    while(my $row=$sth_gt->fetchrow() ){       #Just grab the first matching table
        if($row =~m/$prefix$/){
            $row_to_use=$row;
        }
    }
    print "a row is $row_to_use\n";
    my $sth_gc=$dbh_g->prepare("SHOW COLUMNS FROM $row_to_use");
    $sth_gc->execute or die "SQL Error: $DBI::errstr\n";

    while(my @col_name=$sth_gc->fetchrow_array()){
            $response="";
            while( $response ne "y" && $response ne "n"){
                print "$col_name[0], $col_name[1]? [Y\\n] ";
                $response=<STDIN>;
                chomp($response);
                if ($response eq ""){
                    $response = "y";
                    print "y";
                }
            }
            if($response eq "y"){
                push(@c_names,$col_name[0]);
                push(@c_types,$col_name[1]);
            }
```

```
    }

    print "#Using " . @c_names . " columns for arff.";
    for(my $t=0; $t<@c_types; $t++){
        if(lc $c_types[$t]=~m/int|float|decimal|double/){
            $c_types[$t]="numeric"
        }
        elsif(lc $c_types[$t]=~m/char/){
            $c_types[$t]="string";
        }
        elsif(lc $c_types[$t] eq "date"){
            $c_types[$t]="date";  #nochange
        }
        else{
            print "WARNING: Not sure how to deal with \"$c_types[$t]\", using type
                \"string\" for .arff\n";
            $c_types[$t]="string";
        }

        if(lc $c_names[$t] eq "attack_score"){
            $c_types[$t] = "{0,1}";
        }
        if(lc $c_names[$t] eq "attack_name"){
            #Trim off the crap from Testing data?
            my @att = keys(%attacks);
            my $tmp=join(",",@att);
            $c_types[$t] = "{$tmp}";
        }
    }
    print "Please enter name for config file- ";
    while($outputfile eq ""){
        $outputfile=<STDIN>;
        chomp($response);
    }

    print "Writing config to $outputfile\n";
    open(OUTFILE, ">$outputfile");
    for(my $i=0; $i<@c_names; $i++){
        print OUTFILE "$c_names[$i],$c_types[$i]\n";
    }
    close(OUTFILE);

    $response="";
    while($response eq ""){
        print "Continue to extract data?[y\\n] ";
        $response=<STDIN>;
        chomp($response);
    }
    if($response eq "n"){
        exit;
    }

}


sub check_tables_for_identicalness{
```

```
    print "#Checking to make sure all tables are identical on prefix \"$prefix\".\n"
        ;
    my %hash_table_cols=();
    my $sth_gt=$dbh_g->prepare("SHOW TABLES");
    $sth_gt->execute() or die "SQL Error: $DBI::errstr\n";
    my $at_least_one=0;
    while(my $row=$sth_gt->fetchrow()){

        #print "matching $row with $prefix\n";
        if($row =~m/$prefix$/){
            $at_least_one=1;
            &count_attacks($row);
            $hash_table_cols{$row}="";
            my $sth_gc=$dbh_g->prepare("SHOW COLUMNS FROM $row");
            $sth_gc->execute or die "SQL Error: $DBI::errstr\n";
            while(my @col_name=$sth_gc->fetchrow_array()){
                $hash_table_cols{$row}=$hash_table_cols{$row}." $col_name[0],
                    $col_name[1]";
            }
        }
        else{
                #print "Ignoring table \"$row\"\n";
                #exit;
        }
    }
    my $tmp= $hash_table_cols{(keys(%hash_table_cols))[0]};
    while ( my ($table, $cols) = each(%hash_table_cols) ) {
        if($tmp ne $cols){
            print "ERROR: Tables do not have identical column structures. Failed on
                comparing \"" . (keys(%hash_table_cols))[0] . "\" with \"$table\".\n
                ";
            exit;
        }
    }
    if($at_least_one==0){
        print "ERROR: \"$prefix\" did not match any tables\n";
        exit;
    }
    print "#Compared ". keys( %hash_table_cols ) . " tables - identical column
        structures. Continuing\n";

}
```