ABSTRACT

Title of Thesis: Stateless Detection of Malicious Traffic: Emphasis on User Privacy

Paul Halvorsen, Master of Science, 2013

Thesis directed by: Dr Anupam Joshi
Department of Computer Science

In order to allow flexibility in deployment location and to preserve user privacy we have performed research into stateless classification of network traffic. Stateless detection allows for flexibility in deployment location because traffic on a network does not necessarily follow the same path to and from the end points. By only requiring a single direction of traffic, we have the ability to deploy this classifier anywhere on a network. We also do not require the data from a packet which preserves user privacy and allows for the classification of encrypted traffic.

Our research shows that it is possible to determine if traffic is malicious by using packets traveling in a single direction and without the data contained in the packet. Our research shows that with the use of the timing of the packets, time to live value, and source and destination IP addresses and ports, it is possible to determine if the traffic is malicious. In this way we are able to deploy the classifier anywhere on a network, preserve user privacy, and classify encrypted traffic.

Stateless Detection of Malicious Traffic: Emphasis on User Privacy

by

Paul Halvorsen

Thesis submitted to the Faculty of the Graduate School of the University
of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Master of Science
2013

Advisory Committee:
Dr. Anupam Joshi, Chair/Adviser
Dr. Charles Nicholas, Committee Member
Dr. Tim Finin, Committee Member

Dedication

Dedicated to my wonderful wife Meg, without her loving support I would never have come so far.

Acknowledgments

First and foremost I would like to thank my adviser Dr. Anupam Joshi. With his support I was able to discover a topic, obtain the data necessarily for testing, and through his knowledge and experience this research was a success. I would also like to thank the rest of my committee, Dr. Charles Nicholas and Dr. Tim Finin, their knowledge and experience helped me to refine my research and my paper. I would like to thank the entire committee for taking the time to communicate with me and review the research that I performed.

I would also like to thank my family and friends. With their love and support I was able to persevere through the stress to complete my master's. I would like to especially thank my wife Meg for all her patience and support throughout this process.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
DNS	Domain Name System
DPI	Deep Packet Inspection
FOSS	Fully Open Source Software
FTP	File Transfer Protocol
HMM	Hidden Markov Model
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
IPS	Intrusion Prevention System
JVM	Java Virtual Machine
LTS	Long Term Solution
MySQL	My Structured Query Language
P2P	Peer To Peer
PII	Personally Identifiable Information
SCP	Secure Copy
SCTP	Stream Control Transmission Protocol
SFTP	Secure File Transfer Protocol
SNMP	Simple Network Management Protocol
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SSL	Secure Socket Layer
SSN	Social Security Number
SVM	State Vector Machine
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
VM	Virtual Machine
WLAN	Wireless Local Area Network

Chapter 1

Introduction

When the Internet was first developed, it was utilized primarily by Universities in order to communicate their research and other academic material. Since there were very few entities attached to the network, there was an assumption of trust among members. As the Internet expanded into people's homes, this trust could no longer be assumed, since anyone could have access. As personal computers and institutions began to be attacked for personal and financial gain, research was needed to detect and mitigate these attacks [17].

Detecting attacks can be performed in two primary locations: on the host and on the network [12]. Anti-virus software focuses on detecting malicious activity on the host by scanning the file system, memory, and registries. Intrusion detection systems (IDS) and intrusion prevention systems (IPS) (further discussed in Section 2.2) detect malicious activity on the network. Our research presented here focuses on discovering malicious activity on the network. To detect malicious traffic, signatures are created using the packet contents and traffic patterns to which the traffic is compared. These signatures limit the detection to only known attacks and cannot learn to detect new attacks. Previous research has been conducted into using learning algorithms to generate these signatures, upon which our research builds.

The two primary ways of detecting malicious traffic are deep packet inspection

(DPI) and traffic analysis. DPI collects data from the contents of the packet and determines if any sub-strings in that data match known malicious activity. DPI will take longer than traffic analysis because of the need to perform string comparisons on approximately 67 to 508 bytes per packet to known patterns [28]. String comparisons have a complexity of the size of the pattern n times the size of the string m, so $O(n * m)$ [1]. There has been research performed to speed up DPI in general [13, 14, 19], by running the string comparisons in parallel [16, 15, 24] and by creating specialized the string comparisons for DPI [11].

DPI has two other flaws in addition to the time it takes to perform the detection. First with needing to analyze the packet data, this could lead to privacy concerns for the end user. People are no longer using the Internet just to consume information, but to produce information through blogs, micro blogs, websites used to share content, communicate with others, store sensitive information, perform banking transactions, and purchase items. Packet data can contain personally identifiable information (PII) such as home address, social security numbers (SSN), credit card numbers, and more. The second issue with DPI is much of the traffic online is now being encrypted, in order to protect PII. It is not possible to analyze the stings of an encrypted packet, since it is impossible to view the contents of an encrypted packet.

Previous research has been performed to focus on traffic patterns rather than DPI, to achieve faster detection and detect malicious encrypted traffic. The previous research detects which applications are sending traffic [2, 14, 20, 30] and if the traffic is malicious [6, 13, 18, 21, 27]. These algorithms, using two way traffic, focused on

specific types of traffic, including HTTP, HTTPS, FTP, SFTP, etc. Our research shows a generic way to detect malicious traffic moving in a single direction without needing DPI.

Our research classifies traffic as malicious or benign by observing all traffic going through a network. The algorithm developed through this research uses a Bayesian Network to learn the patterns of malicious traffic and to classify unknown traffic. The algorithm first collects all traffic flowing through the network, then performs detection using traffic flowing in a single direction. The reason to only observe traffic in one direction is to allow flexibility in deployment location and to reduce the amount of processing that is necessary for classification. Through this technique, this algorithm is able to achieve a true positive rate of 0.9 with a false positive rate of 0.07. These results show that classifying traffic, using only traffic data, and traffic flowing in a single direction, is a viable way of detecting malicious activity.

This thesis will present the findings of the previous research to provide context. It will continue by describing the setup of the network and methods used to perform the research. After presenting the findings, a discussion is given to provide an explanation of the results and a comparison to the previous research. The paper concludes by summarizing the findings and providing future direction for the research.

Chapter 2

Literature Survey

2.1 Networking

Networking is the way in which nodes on a network communicate with one another. A network can be isolated to just a few nodes or be as large as the Internet as a whole. Network traffic flows from one node to the next so that two end points can communicate. Traffic flowing on the network will not necessarily take the same path to and from each of the end points. Each node on the network can advertise a different distance per interface and based on these advertisements, traffic may take different paths.

Each network can be broken into sub-networks which have nodes on the edge of the network. The edge of a network is a location through which all or a portion of the traffic must travel. For effective monitoring with the current research, all of the edge nodes need to be monitored and coordinated through a single IDS or IPS, which is further discussed in Section 2.2. With our research we analyze traffic moving in a single direction, thus will not be restricted to the edge of a network.

Traffic also communicates over several different protocols using a different port per protocol. Some of the research that has already been conducted focuses on specific protocols (Section 2.3.1). Our research collects traffic across all of the protocols. Other research focuses on the data within the packet (Section 2.5), where

our research uses only traffic data and headers to perform classification (Section 2.3.1 and 2.3.3).

2.2 Intrusion Detection Systems

An Intrusion Detection System (IDS) is a set of tools that run passively on a network to determine if traffic is behaving maliciously. By running passively the IDS observes the packets being transferred over the network, but takes no action against the flow of traffic or the hosts on the network. In some cases passive collection is preferable to active because passive collection will not bring awareness to the attacker that they are being observed. This way a greater amount of data can be collected regarding the techniques and exploits an attacker is using to enter the system. Passive collection will also not affect the end user when false positives are flagged.

Active systems are referred to as Intrusion Prevention Systems (IPSs), which are generally used in conjunction with an IDS. IPSs seek to stop the malicious traffic before it can do any harm, which has the advantage of stopping an attack from happening. Unfortunately as soon as an attacker realizes they are being blocked they will switch tactics and try other attacks. This research will focus on a passive IDS, which can later be integrated into an IPS.

There are several commercially available IDS systems, both closed source proprietary solutions and fully open source software (FOSS) solutions. The most widely used FOSS IDS is *Snort* [25], which is developed and maintained by *Sourcefire*. Snort

utilizes signatures developed by the community or custom created by the user to detect and prevent attacks on a system. Signatures used by Snort as compared to our research is discussed in Section 2.3. Snort observes packets from the edge of a network so that it will be able to observe all traffic entering and exiting the network. Snort performs not only passive observation, but when configured to do so, will proactively shut off flows and block IP addresses that trigger the signatures.

Another IDS, used for research rather than commercial use, is called *CoralReef* [13]. CoralReef is an architecture that passively collects and captures traffic flowing through the edge of a network. Using CoralReef, an end user is able to build their own modules to interact with the application programming interface (API) to access the data. With this method an end user can create any signature or learning algorithm that is necessary to their particular application. CoralReef also comes with built in functionality to convert traffic collection to other file formats, analyze timestamps, collect DNS usage statistics, and more to aid in detection.

Current widely used IDS systems use signatures to identify traffic flows. These signatures are very specific and will miss new attack patterns and new vulnerabilities. Using a learning algorithm, an IDS has the ability to dynamically alter its detection such that it will detect new attacks faster.

2.3 Signatures

IDSs use what is known as signatures to detect malicious activities. These signatures are a set of defined rules to classify traffic as malicious or benign. Most

signatures are static and will only look for a very specific pattern in the traffic (i.e. a packet every three seconds), but will miss anything not in that specific signature (i.e. a packet every five seconds). Due to this limitation a large number of signatures are needed to block a wide range of malicious traffic. Snort comes with 16,000 signatures by default and currently has over 20,000 additional signatures for download [25]. Our research reduces the number of items that need to be stored and compared against since it uses machine learning to produce a single signature.

There are two primary kinds of signatures, traffic and data signatures, each observing different parts of the packet. Traffic signatures observe patterns in the flow of traffic (Section 2.3.1), on which our research focuses. Traffic analysis will not infringe on user privacy and can detect patterns in encrypted packets (Section 2.5). Data signatures perform DPI, which observes data inside the packet to determine if malicious code is being (Section 2.3.3). This could infringe on user privacy since the packet data is where user names, passwords, the contents of emails, and other forms of communication are located.

2.3.1 Traffic Signatures

Traffic signatures observe patterns in the flow of traffic using information contained in the IP header and the time the packets are observed. From the packet header the ports, IP addresses, and time to live (TTL) values are gathered. Ports and IP addresses are used to identify a *session* which is a grouping of packets that are sent or received by the same application. Throughout the previous research,

there have been over 40 different flow indicators. These indicators can be broken into five categories: header size, data size, number of packets, timing of the arrival of packets, and other [14, 6, 30, 27, 18, 2].

In their research Wright et al. (2006) observe traffic patterns to determine which protocols are being used: FTP, HTTP, HTTPS, SSH, SMTP, etc. They were able to achieve an average true positive detection rate of 99.66% with an average false positive rate of 1.2% [27]. For our research we created a generic detection algorithm, that does not focus on any set of protocols. This will reduce accuracy because we are attempting to classify a wider range of traffic.

In the research performed by Zhang et al. (2011), they achieved an average detection rate of 92.26% with an average false positive rate of 1.29% [30]. Their research focused on detecting malicious activity on a wireless network. They achieved this by collecting data passively on a wireless local area network (WLAN) for 60 seconds and applying a Hidden Markov Model (HMM). They did this while correlating incoming and outgoing packets to form a stream. Our research utilizes a Bayesian Network to perform classification and applies this to a *session* (Section 2.5) of packets moving in a single direction.

Both Auld et al. (2007) and Li et al. (2007) present research that classifies traffic using stateful collection and classification [2, 18]. Stateful, in this context, refers to the fact that the sessions consist of both inbound and outbound connections, where stateless connections refer to only a single direction. Auld et al. (2006), using a Bayesian Neural Network and 28 classifiers, were able to achieve an average detection rate of 96.6% with an average false positive rate of 1.02%. Li et al. (2007),

using support vector machines (SVMs) and nine classifiers, were able to achieve an average detection rate of 96.92% and an average false positive rate of 6.59%.

The research that we have performed focuses on traffic signatures. By observing only the traffic, the privacy of users will be preserved since the data in the packet will not be observed or stored. The current techniques that only use traffic signatures are *stateful* detectors, where our research is *stateless*. Stateful detection utilizes traffic flowing in both directions to perform classification, stateless detection utilizes traffic flowing in a single direction. This will be needed for detectors that are not placed at the edge of a network, or a network with multiple entry points. The stateless is important because traffic does not always flow in the same direction across a network.

2.3.2 Visual Signatures

Some research has also gone into visualizing traffic [6]. This research, conducted by Leo Breiman et al. (2001), was performed under the assumption that a person is able to see patterns and anomalies better than a computer. While humans do have a better capability of observing patterns, human intervention will only slow the process of locating abnormalities. Detection will be slower because a computer can process information faster than a human. Visualizing traffic does have merit because once a human finds an abnormality, the abnormality can be translated into a signature utilized by an IDS.

2.3.3 Data Signatures

Data signatures, which use DPI, observe the entire packet content including the IP header, protocol header, and packet payload. The reason for performing DPI is to detect malicious code contained within the packet payload. This has an advantage over traffic signatures since some attacks can hide by mimicking benign traffic. However, DPI requires a greater amount of processing than traffic signatures and thus can take a longer amount of time to complete. To perform DPI, the data first needs to be collected, then string comparisons take place. As stated, this type of inspection could lead to privacy concerns and is ineffective against encrypted traffic.

Data inspection research has been focused on improving pattern matching for strings. Kumar et al. (2006) presented research on how to speed up asynchronous regular expression comparisons [16, 15]. Kefu et al. (2008) continued this research by improving the speed of pattern matching in strings for synchronous look-ups [11]. Liao et al. (2012) rearranged and developed additional DPI rules that eliminated redundant or obsolete searches and made the overall search more efficient [19].

2.4 Limitations of Current Research

The current techniques are limited because they look for very specific signatures that will leave out new attacks and can be easily overcome. New attacks will not be picked up since a signature has not been developed for the new pattern. Attacks can also be tailored to either look like legitimate traffic or to work around

these signatures. Once an attacker learns how a signature is setup, they will be able to tailor their attacks to circumvent that signature.

The current research is also restricted to the edge of a network because they require both incoming and outgoing traffic to be correlated before performing analysis. On the global network, depending on the route advertisements of network nodes, traffic will not necessarily take the same path between endpoints. In order to have the flexibility to place the detection anywhere on a network, stateless detection is needed. Stateless detection only requires traffic flowing in a single direction.

The other issue with the current state of research is that it performs DPI which can infringe on user privacy and is ineffective against encrypted traffic. The data portion of a packet contains private information that can include but is not limited to user names, passwords, email content, credit card information, and other PII. Users of the network also encrypt packets, making it impossible to inspect the packet contents, causing DPI to fail. Encryption has become wide spread in order to protect PII and renders the data contained in the packet unreadable to anyone other than the client and server.

2.5 Proposed Research

With privacy concerns increasing and cyber attacks becoming more prevalent, there is a need to seek a method to keep networks secure, while protecting the users privacy. Our research detects an attack while respecting the privacy of the user by not inspecting the packet contents. Current intrusion detection techniques focus

on specific kinds of attacks, for example botnets, fuzzers, or DNS anomalies. Also, current IDS systems observe the packet data, possibly infringing on user privacy.

For this research, we have created a generic detection system by using only the time the packets are observed and the packet headers. By not observing the packet data, our research will protect user privacy, because the packet data is where the sensitive information, such as PII. The drawback to this technique is that without the packet data, this could lead to decreased detection rates as malicious traffic can mimic the patterns of benign traffic. This research provides insight into how accurate a detection system can be without inspecting the packet data.

The other piece to our research is generic intrusion detection. As stated, current IDS systems detect very specific forms of attack, such as botnets (a network of hosts who's owners don't know they are being used) or fuzzers (a technique where random input is given to a program or host to try and make it crash). Our research brings together several different techniques in order to detect several forms of attacks. Each of these techniques are used in conjunction with one another to produce a single classification. Generic detection can also lead to lower accuracy rates because we are attempting to classify a wider range of data. Our research simply states if an attack is occurring and does not give a confidence value or type of attack.

To perform this detection, we have implemented a stateless classifier that only uses the IP headers. The reason for stateless detection is to allow flexibility of deployment location. Our classifier will be able to perform classification at any location on a network. The parts of the traffic used to perform generic detection are

the number of unique ports being accessed, the number of lower value ports being accessed, how periodic the packets are, how rapid the packets are being observed, the number of packets within a session, the average TTL, and the standard deviation of TTL values. This is the first time these indicators have been brought together using learning algorithms.

Our research has produced a proof of concept collector and classifier. It utilizes a closed network of several virtual machines (VMs) with the host running the collector and classifier. Greater detail of the setup is given in Chapter 3 and results are given and discussed in Chapter 4.

Chapter 3

Methods and Materials

3.1 Network Setup

The network used in this research was made up of several VMs and two hosts, with one of the hosts performing monitoring. The VMs utilized Ubuntu 12.04 and Debian 6.0.5 and were hosted using Virtual Box. The host for the VMs was Ubuntu 12.04 and Windows 7. The VMs were used to send and receive the traffic and the Ubuntu host was used to monitor and classify the traffic. Benign traffic was sent by some of the VMs and obtained from browsing known benign websites, opening an SSH session (secure shell, used to established remote command line access), and playing games.

3.2 Simulated Traffic

Malicious and benign traffic were simulated through several different tools. The malicious traffic was simulated using *tcpreplay* [26] and a *Backtrack VM* [3] (Section 3.2.1). The benign traffic was also simulated using tcpreplay and by performing normal tasks that involve sending and receiving packets (Section 3.2.2). All of this traffic was collected by the Ubuntu 12.04 machine that was hosting several of the VMs. The traffic was then processed to determine if the traffic was malicious.

For both the simulated malicious and benign traffic, tcpreplay was automated using python scripts (Section 3.2.3).

3.2.1 Malicious Traffic

To send malicious traffic tcpreplay and Backtrack were used. Tcpreplay is a tool that will send packets described in the standard *pcap* format. The standard pcap file format, is a format that describes packets collected through tools such as *wireshark* and tcpdump. These files give the full packet details, including all parts of the IP header, protocol header, and data [8, 9, 22]. To make sure that these would send packets to and from the correct IP address, two other tools were used: *tcpprep* and *tcprewrite*.

Tcpprep determines which IP addresses, in the pcap file, were assigned to the server and which ones were assigned to the client for each session. It creates a cache file to describe the assigned IP addresses and to help decrease the time required to send the packets. Once the server and client IP addresses are determined, tcprewrite will create a new pcap file with new client and server IP addresses. This new pcap file is then used by tcpreplay to send (or replay) the packets to the new client from the new server. In our research the source IPs became those of the attack VMs and the client IPs were three different client VMs.

The pcap files used for the malicious data were downloaded from several collections of traces from attacks against honeypots. The different types of attacks used were: buffer overflow, DNS remote shell, fragmentation, ping tunnel attack,

TCP scans, UDP scans, SMB attack, teardrop attack, ICMP echo fingerprinting, ICMP TCP scan, ICMP UDP scan, several xprobe ICMP attacks, and a zlip attack. For training purposes, each of these attacks were sent from a different IP address.

Another tool used to send malicious traffic to the clients, was a *Backtrack 5 VM*. Backtrack is a VM built on Ubuntu 10.04 LTS and is used for penetration testing (pen testing). To perform pen testing, Backtrack uses a tool known as *Metasploit* with the graphical front end *Armitage*. For this research, Armitage was used to perform scanning of the network to discover the hosts, then to send several attacks. These attacks include: hailmary (uses all available exploits), HTTP, samba, webapp, ssh, and wyse exploits.

We used specific exploits from each of the above categories to test the detection. Of the HTTP exploits available, activecollab_chat, apprain_upload_exec, contentkeeperweb_mimencode, cuteflow_upload_exec, and ddwrt_cgibin_exec were used. Of the samba exploits available, trans2open and usermap_script were used. Of the available webapp exploits, cakephp_cache_corruptoin, guestbook_ssi_exec, openview_connected_nodes_exec, and twiki_history were used. Of the available ssh exploits f5_bigip_known_privkey was used and of the available wyse exploits hagent_untrusted_hsdata was used.

3.2.2 Benign Traffic

Benign traffic was simulated through the use of tcpreplay and manual activities. Tcpreplay was used in the same manner as it was for malicious traffic, utilizing

pcap files downloaded from different sources [8, 9, 22]. The benign pcap files were traces of bittorent, HTTP, ICMP, IPMI, SCTP, SNMP, TCP, and telnet. Each of these were sent from a different IP address.

Also used for benign traffic was manual interaction with other servers and services. An ssh session was opened to a remote server where sever commands were run to navigate through the directory structure. An SCP (secure copy, used to copy files from one host to another) session was started and 84 files were transferred. Several (around 50) web pages were manually navigated to and browsed. During web browsing music was streamed from *www.pandora.com*, videos were streamed from *www.hulu.com*, *www.foxnews.com*, and *www.cbs.com*; web mail was used from *www.gmail.com* and *www.yahoomail.com*; images and animated images called gifs were viewed on *www.imgur.com*; dynamic content was viewed from *www.facebook.com*, *www.twitter.com*, and *www.slashdot.com*; and games were played on *www.facebook.com* and *www.games.com*. This list is just some of the websites visited and is not exhaustive.

3.2.3 Automating Tcpreplay

The traffic being simulated via tcpreplay was automated with two python scripts and a configuration file. The first python script prepared the new pcap files by altering the IP addresses to those used for the test and the second replayed the packets described in the new pcap files. The configuration file provided information on which pcap files to use and from which IP addresses to send the packets. Since

tcpreplay can simulate traffic from multiple IP addresses while residing on a single host, each VM was responsible for sending multiple malicious flows.

3.3 Collection

To collect the traffic, *C++* with a *MySQL* database was utilized. C++ was chosen over Java and scripting languages, because it runs naively after compilation. Java compiles to byte code and runs in a VM and scripts are interpreted at run time, causing both to run slower. C++ was also chosen due to its ability to manipulate memory, allowing for fast parsing of the incoming packets. Fast parsing is necessary to keep up with the volume of traffic on a network. C++ was chosen over *C* for its ability to manipulate strings.

In order to collect the traffic with C++, *libpcap* was utilized. Libpcap is the same library used by wireshark and other traffic collection programs to provide the raw bites for all traffic on a given interface. The ability of C++ to manipulate memory is utilized to retrieve the necessary parts of the packet. Using *libmysql-cppconn*, the packet information is added to the MySQL database to be saved for further processing. The collection for this research only collects the time the packet is observed, TTL, source and destination IP addresses, and the port numbers from the packet headers.

For this research we collected 202,047 packets. This was broken into 22,295 sessions by dividing the packets by source IP, destination IP, source port, and by a division of at least 60 seconds between packets. These sessions were then divided

into benign and malicious sessions. There were 20,843 benign sessions or %93.5 of the traffic and 1,452 malicious sessions or %6.5 of the traffic.

3.4 Classification

The learning and classification of the traffic is done using *Java* and *WEKA*. WEKA is an artificial intelligence (AI) training tool, that takes formatted data files to train a number of AI algorithms used for classifying data. WEKA has an interface that allows a developer to write Java in order to train and classify data. This interface is the reason Java was chosen. The leaning technique chosen for our research was a Bayesian Network.

The Bayesian Network is an acyclic graph (a graph with no cycles) that determines the probability of a particular outcome. Each edge in the graph represents a probability that the next event will occur. Once training is complete, the Bayesian network can determine the probability of several outcomes based on the input. In the case of this research, given the signatures, the algorithm will produce a probability that the traffic is malicious or benign.

3.4.1 Signatures

Three primary indicators were used for learning and analyzing the traffic: the ports used, the time to live (TTL) values, and the time the packets were observed. For the ports, the number of unique ports and the number of unique ports below 1024 (the reserved ports) were calculated. For legitimate traffic, the average number

ports observed tends to be around two. When connecting to a legitimate service, the service tends to negotiate a second port for use in order to perform load balancing. With malicious traffic, in regards to ports, there are two primary scenarios, a directed attack and a scanner. With a directed attack, only a single port will be observed. Attacks will target a specific service on a remote machine, and thus is required to use the one port on which the remote service is listening. With scanners, more than two ports will be observed, since the scanner is attempting to locate vulnerable open ports.

TTL values are checked for anomalous patterns as well. When analyzing the TTL, the number of values below 50, the number of unique values, the mean value, and the standard deviation (a unique value for our research) of the TTL values are calculated. The idea here is that most applications set the TTL value to the highest possible value of 255. Other than mapping a network, there are very few applications where a low TTL is an advantage. The reason that legitimate traffic would not use low TTL is so that it won't drop packets before reaching the client. Malicious users will use low TTL values in an attempt to mimic legitimate traffic or to map a target network.

The time the packets were observed is also collected. Here the number of packets within a given amount of time, the average number of packets per second, the total time a session was open, the mean time between packets, and the standard deviation of the time between packets (a unique value for our research) were calculated. Legitimate traffic tends to be more random with greater time between packets and malicious traffic will potentially be concentrated into bursts of traffic

close together. Malicious traffic tends to be concentrated because in order to exploit most vulnerabilities a burst of packets is needed. A malicious user may also want to get the payload uploaded as quickly as possible before their traffic can be blocked or the system patched. Legitimate traffic tends to be slower and more random, since it is generated by a person.

After all the data has been classified, python was used to graph the results. When WEKA runs a test it produces the number of false positives, true positives, false negatives, and true negatives for varying degrees of accuracy. Using the *matlib-plot* library available for Python and the output from WEKA, graphs were produced to compare the false positive rates and true positive rates (Chapter 4)

Chapter 4

Results and Discussions

4.1 Traffic Sessions

After collection the traffic was divided into sessions which were classified as either malicious or benign. These sessions were determined by three factors: the source and destination IP addresses, the source port, and the timing of the packet. Only the source port is used to determine sessions since usually applications will run on a single port, but may be routed to multiple ports on a destination for load balancing. An attacker will also touch multiple ports as the attempt to scan a remote host for vulnerabilities. The start and end of sessions are also determined if there is at least 60 seconds between a sent or received packet. If there are no ports available, as in ICMP traffic, then the IP address and timing of the packet are used to determine a session.

Our research focuses on *stateless* detection. Stateful detection determines sessions by using traffic that goes both to and from a host. With stateless detection, this algorithm only needs to observe one way traffic saving computational time. This is advantageous because traffic does not necessarily take the same path through the network. Packets will take different paths depending on the advertised routes from nodes along the network. If a network sensor was placed along the route it would only be able to observe traffic flowing in a single direction.

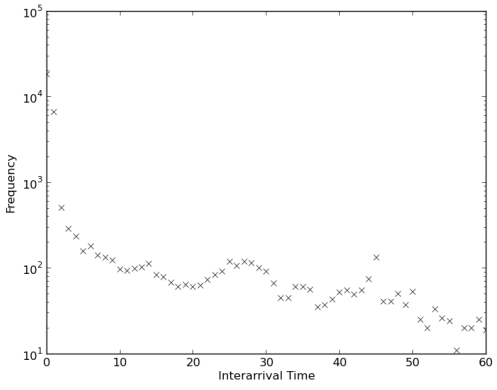

Figure 4.1: Benign Interarrival Time

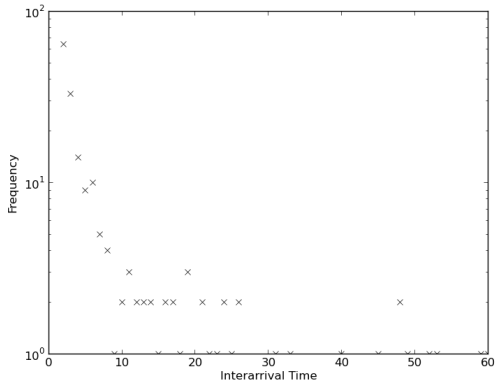

Figure 4.2: Malicious Interarrival Time

For our research we collected 202,047 packets that was comprised of 20,843 benign sessions and 1,452 malicious sessions. Figure 4.1 shows the frequency of the interarrival times in seconds for benign traffic. This graph is on a logarithmic scale to better visualize the traffic. These interarrival times are comparable to those discovered by Barakat et al. [4] showing that our traffic is comparable to real world traffic. Figure 4.2 shows the frequency of interarrival times for malicious traffic, also on a logarithmic scale. These two figures show the difference between benign and malicious traffic.

4.2 Individual Classifiers

The three pieces of data used by this research to classify traffic was the destination port number, the TTL value, and the time the packet was observed. From these data points several values, described in Sections 4.2.1, 4.2.2, and 4.2.3, were derived. Each data point was tested individually to assess it's significance before combining all three. To compare each of the classifiers, a false positive rate of 0.1

will be used. When discussing the quality of each classifier, the largest true positive rate with corresponding false positive rate below 1 will be noted.

4.2.1 Ports

After gathering the destination ports, this algorithm determines the number of unique ports and the number of unique reserved ports, values below 1024, for each session. The ports are used to detect an attack targeting specific applications or scanners attempting to locate vulnerable applications.

The two cases we observed while performing classification were scanners and directed attacks. Scanners will cause a larger number of unique ports, however scanners are less frequently used since they are noisy. Directed attacks were observed more often which displayed a lower number of unique ports. Malicious traffic will target a single port since each application will listen on a single port at a time. This will cause the average number of unique ports observed for malicious traffic to be lower.

Between malicious and benign traffic the number of unique ports is fairly equal at 1.004 unique ports for malicious traffic and 1.338 unique ports for benign traffic. Benign traffic also had a greater number of unique ports below port number 1024 at an average of 0.611 unique ports for benign traffic and 0.187 unique ports for malicious traffic.

The results for using only ports as the data point for classification are displayed in Figure 4.3. Using just the ports as an indicator, high true positive rates are only

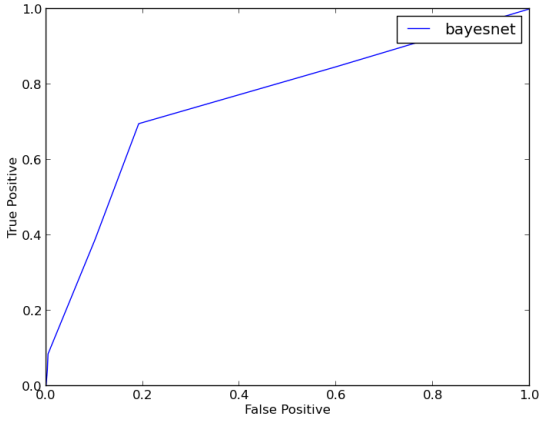

Figure 4.3: Port Only Classification

possible with high false positive rates. This is noted when the highest observed true positive rate of 0.846 only occurs when there is a false positive rate of 0.598. When reducing the false positive rate to acceptable levels the true positive rate also drops significantly. If the false positive rates are reduced to below 0.100 the true positive rates are reduced to below 0.388. This means that over 60% of the attacks are missed.

4.2.2 Packet Timing

After collecting the times the packets were observed several other pieces of information were calculated. The average time between packets, the number of packets observed in the connection, and the duration of the session lasted for were calculated. The standard deviation of the time between packets, which is a unique value to our research was also calculated. Using this data produced a better detection rate than the data gathered from the ports.

The data calculated from packet times are effective because the initial attacks tend to be very short lived bursts of traffic going in one direction. Previously it was stated that this research focuses on stateless detection, which is significant since in some cases benign traffic will periodically wait for acknowledgments which increases the time between observed packets. The average time between benign traffic was 6.156 seconds and the average time between malicious traffic was 0.324 seconds. Benign traffic sessions also tend to be longer and contain more packets.

Malicious traffic also tends to have a more regular interval between packets. The average standard deviation for malicious traffic is 0.209 seconds and the average standard deviation of benign traffic is 3.334 seconds. Malicious traffic tends to be more regular for two reasons. The first is the attacker will send a large amount of traffic all at once to exploit a target as fast as possible. The second is that if a host is calling back to an attacker, the traffic involved tends to be fairly evenly spaced so the attack knows when the host will call back. Benign traffic tends to be more random because it is determined by human interaction.

Using only the time the packets are observed for classification, there is a noticeable increase in accuracy over the ports. These results are displayed in Figure 4.4. To achieve the highest observed true positive rates of 0.906, the false positive rates are 0.786. With a false positive rate below 0.100, the true positive rate is reduced to 0.521, which is better than the port only rate of 0.388.

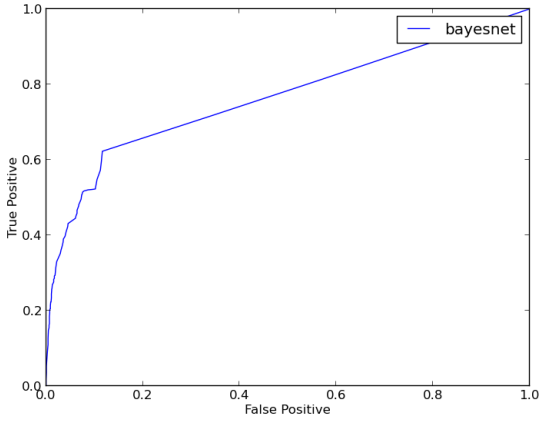

Figure 4.4: Timing Only Classification

4.2.3 Time To Live

The TTL values showed the greatest accuracy in detection among the three data points collected. After collecting the TTL values, the algorithm calculated the number of unique TTL values, the average TTL value, and the number of TTL values below 20 hops. Looking at the raw data, the only calculated data point that was significant was the average TTL value. The majority of the sessions only had a single TTL value used, thus the number of unique values was one. Simulated malicious traffic for this research contained an average TTL value of 64.647 hops, while benign traffic had an average of 72.805 hops.

The TTL values continue to show improvements over the other two classifiers. This is different than what we originally predicted before running the tests, this is further discussed in Section 5.3. These results are shown in Figure 4.5. To achieve the highest true positive rate observed of 0.972 there is a false positive rate of 0.671. To compare with the other classifiers, achieving a false positive rate of less than

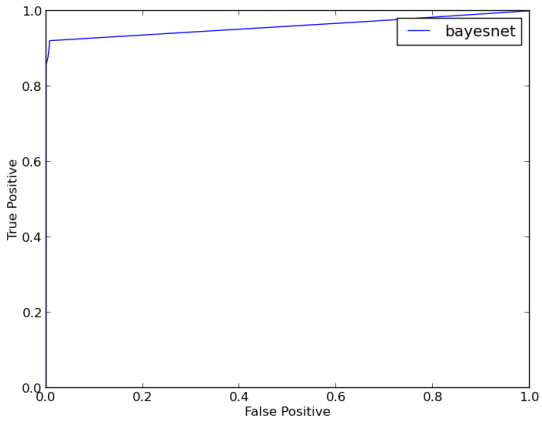

Figure 4.5: TTL Only Classification

0.100 the true positive rate is reduced to 0.921, which is higher than the ports true positive rate of 0.388 and the packet times true positive rate of 0.521.

4.3 Combined Classifiers

Putting all three classifiers together yields the greatest accuracy for classification. Each classifier provides values that are unique to different types of attacks and different types of benign traffic. By putting together the values in each of these classifiers, a more specific signature is created for the traffic, and the more specific the signature the greater the accuracy.

After combining all three classifiers, there is a significant increase above the individual classifiers. The ROC curve of the results are displayed in Figure 4.6 and the confusion matrix for the results are displayed in Table 4.1. To achieve the highest true positive rate observed of 0.999, the false positive rate is 0.742. In comparison to the other classifiers, to achieve a false positive rate less than 0.100, the true positive

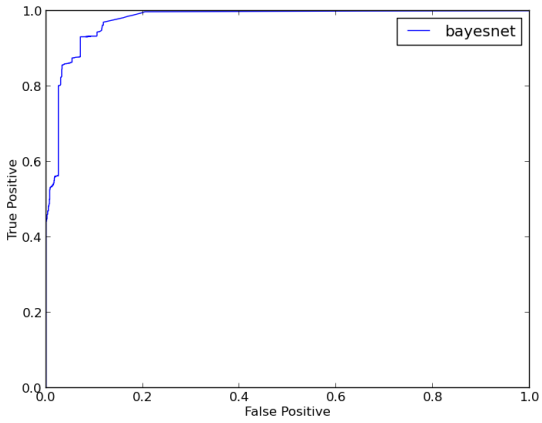

Figure 4.6: Port, Timing, and TTL Classification

Confusion Matrix			
		Predicted	
		Negative	Positive
Actual	Negative	1349	103
	Positive	2076	18767

Table 4.1: Full Classification Confusion Matrix

rate is 0.933. This is greater than the 0.388, 0.521, and 0.921 previously observed. Two months later we ran the classification algorithm again without retraining. For this test we ran six attacks and the updated hail mary attack. As expected, the classification algorithm did not perform as well with the new attacks. After two months, we achieved a true positive rate of 0.805 and a false positive rate of 0.071. These results are discussed further in Chapter 5

Chapter 5

Conclusion

5.1 Comparison

Our research shows that it is possible to detect malicious activity by observing traffic flowing in only on direction. It also shows that it is possible to perform this detection without having to perform deep packet inspection or violate user privacy. Our algorithm is ideal as a pre-filter before performing greater and more thorough inspection. While maintaining a true positive rate of 0.9 it achieved a false positive rate of 0.07.

Our new algorithm, while achieving a comparable true positive rate, had a greater false positive rate than other similar research. Our algorithm has a higher false positive rate because it performs classification on traffic moving in a single direction, without violating user privacy and does so generically. The single direction, or stateless, detection is important, since over a large network, traffic will not necessarily follow the same path to and from an endpoint. By being stateless, our algorithm can perform detection anywhere on a network or the Internet and does not need to be placed on the edge of a sub-network.

Due to the use of stateless and generic detection, our algorithm has greater false positive rates as compared to other algorithms proposed. The algorithm that we developed uses less data than the previous techniques discussed because it performs

stateless detection while maintaining user privacy. With stateless detection, we only observe traffic flowing in a single direction, thus we only collect approximately half the number of packets per session. This makes the algorithm more versatile as it can be placed anywhere on a network where the other algorithms must be placed at the edge of a network.

Compared to the other algorithms proposed, our algorithm is more generic. Several of the other proposed techniques observe specific protocols or search for specific types of attacks. The generic nature of this algorithm makes it less accurate, but it also makes it more versatile. Our algorithm is not restricted to specific types of attacks or specific protocols.

Wright et al. (2006), with their algorithm achieved a true positive rate of 99.66% and false positive rate of 1.2% [27]. In comparison to our algorithm, when achieving a true positive rate of 99.4% it has a false positive rate of 19.6%. While we had a much higher false positive rate, Write et al. (2006) only focus on six protocols: SMTP, HTTP, SSL, FTP, SSH, and TELNET. The new algorithm detects activity across all protocols, which decreases the accuracy.

Zhang et al. (2011) were able to achieve a true positive rate of 92.26% and a false positive rate of 1.29% [30]. Our algorithm achieved a true positive rate of 92.45% and a false positive rate of 7.09%. The research performed by Zhang et al. (2011) closely mimics our research since they attempt to classify user activity over encrypted channels. Using limited information of only traffic patterns, they classify if a user is browsing the web, chatting, gaming, downloading, uploading, watching videos, or using p2p (peer to peer) applications. Our research similarly

uses limited information, but instead of classifying differences in specific benign traffic, we generically differentiate between malicious and benign traffic.

5.2 Summing Up

The research presented here shows that it is possible to differentiate between malicious and benign traffic by collecting traffic flowing in one direction, and observing only the packet headers, and time the packets were observed. Our algorithm has achieved a true positive rate of 0.9 while maintaining a false positive rate of 0.07. While not as accurate as others presented here, this research shows the viability of generic stateless detection.

In its current state, our algorithm can be used as a pre-filter before performing more precise analysis on the traffic. Even using this algorithm as a pre-filter will aid in protecting privacy, as the majority of traffic is correctly marked. This algorithm needs to be researched and developed further in order to achieve a higher accuracy, before becoming a stand alone product.

5.3 Shortcomings

Attackers have several ways to try and evade the classification. First, for an attacker to evade the TTL data, they need to mimic benign TTL values. Typically benign traffic has a higher value for TTL since they tend to set it at the max of 255. The TTL value in a packet decreases with each hop along the network so the attacker would need to discover the average number of hops from the most used web

sites to the victim. This way an attacker will be able to tailor the initial TTL value to mimic normal traffic. While difficult, making this alteration would not hinder the attack, it would require time to figure out the appropriate starting value.

The attacker would be able to get around the data collected from ports by slowing down the scans or by mimicking benign traffic. Slowing down the scans to only one to two ports per minute would not alert our algorithm since it breaks up sessions with a 60 second interval. By slowing down the scan, scanning all 65,535 ports on a single host would take around 22.75 days. If the attacker knows a select number of ports to scan they could reduce this time, but every two ports the attacker wants to scan will take one minute. Due to the length of time to perform scans, this becomes impractical for an attack who doesn't already know their target.

Another way for an attacker to get around the port data is to mimic benign traffic. On average benign traffic uses 1.338 unique ports per session, so malicious traffic would need to use more than one port at a time. An attacker will be able to do this once they are on a victim box, but to perform the attack, the attacker will need to focus on the vulnerable port. So for an attacker, they will not be able to hide an initial attack, but once on a victim, the attacker will be able to hide their presence by having the malware call home on different ports.

The attacker can hide from the data collected for the times the packets were observed by mimicking benign traffic. To do this the attacker will need to slow the attacks to fall within the average time between packets of benign traffic. By increasing the time between packets the time it takes to complete an attack will increase, which will decrease the effectiveness of the attack. Some attacks need

to happen quickly and thus cannot be used, and those that can be used will take more time. The attacker will also have to vary the length of time between packets. Varying the length between packets isn't difficult and will not hamper an attack.

5.4 Future Research

This research can be improved through the use of more data points and faster algorithms. Our research was used as a proof of concept, thus was not optimized for time. To prove that it was possible to be able to detect malicious traffic while protecting user privacy, we used WEKA as a simple solution to implement the learning algorithm. WEKA runs in the Java Virtual Machine (JVM) which is slower than native code and is a general purpose program that has not been optimized for any one algorithm. By using native code, such as C or C++, and by optimizing the algorithm toward the specific AI component the algorithm will be able to run much faster.

Another way to speed up the algorithm is through concentrated and combined queries. Our proof of concept queries an entire time period of packets, no matter the source or destination IP address, or the ports involved. By ignoring these factors, a larger number of rows are queried for learning and classification than are necessary. By limiting the number of rows queried, less processing needs to be performed before learning and classification.

Our proof of concept also queries and calculates each classifier individually. Many of these calculations perform similar or identical queries and calculations and

can thus be combined. By combining all three data points into a single query and set of calculations the time it takes to perform the calculations will be reduced by a factor of three.

Discovering new data points to collect and calculate will also improve this algorithm by increasing the accuracy. The proof of concept showed that it is possible to perform stateless classification without invading the users privacy and on encrypted traffic, but it still produces high false positives. By discovering more data points, our algorithm's accuracy could be improved to reduce the number of false positives. Currently this algorithm is ideal for filtering before more precise classification can take place. With more data points, this algorithm's accuracy could be improved to be the primary classifier.

By continuing the research to decrease the processing time and increase the accuracy, our algorithm will be able to perform full classification of traffic. Improving the speed can be done by making more specified queries, combining classifier calculations, and by running native code. The accuracy can be improved by discovering additional data points that are restricted to the IP header and the flow of traffic.

Bibliography

[1] Lloyd Allison. Strings. http://www.csse.monash.edu.au/ lloyd/tildeAlgDS/Strings/, 2012.

[2] T. Auld, A.W. Moore, and S.F. Gull. Bayesian neural networks for internet traffic classification. *Neural Networks, IEEE Transactions*, 18(1):223 –239, jan. 2007.

[3] BackTrack. Backtrack linux. http://www.backtrack-linux.org/, 2011.

[4] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, and P. Owezarski. Modeling internet backbone traffic at the flow level. *Signal Processing, IEEE Transaction*, 51(8):2111 – 2124, aug. 2003.

[5] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. 10.1023/A:1010933404324.

[6] Gregory Conti and Kulsoom Abdullah. Passive visual fingerprinting of network attack tools. In *Proceedings of the 2004 ACM workshop on visualization and data mining for computer security*, VizSEC/DMSEC '04, pages 45–54, New York, NY, USA, 2004. ACM.

[7] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *SIGCOMM Comput. Commun. Rev.*, 37(1):5–16, January 2007.

[8] Evil Fingers. Evilfingers pcaps. https://www.evilfingers.com/, 2010.

[9] Guy Harris. Wireshark sample captures. http://wiki.wireshark.org/SampleCaptures/, 2012.

[10] Geoff Huston. Tcp - how it works. http://www.potaroo.net/ispcol/2004-07/2004-07-isp.htm, 2004.

[11] Xu Kefu, Qi Deyu, Qian Zhengping, and Zheng Weiping. Fast dynamic pattern matching for deep packet inspection. In *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference*, pages 802 –807, april 2008.

[12] R.A. Kemmerer and G. Vigna. Intrusion detection: A brief history and overview. *Computer*, 35(4):27 –30, apr 2002.

[13] Ken Keys, David Moore, Ryan Koga, Edouard Lagache, Michael Tesch, and K Claffy. The architecture of coralreef: An internet traffic monitoring software suite. In *PAM2001, Workshop on Passive and Active Measurements, RIPE*, 2001.

[14] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: Myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 11:1–11:12, New York, NY, USA, 2008. ACM.

[15] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. *SIGCOMM Comput. Commun. Rev.*, 36(4):339–350, August 2006.

[16] Sailesh Kumar, Jonathan Turner, and John Williams. Advanced algorithms for fast and scalable deep packet inspection. In *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, ANCS '06, pages 81–92, New York, NY, USA, 2006. ACM.

[17] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Lawrence G. Roberts, and Stephen S. Wolff. The past and future history of the internet. *Commun. ACM*, 40(2):102–108, February 1997.

[18] Zhu Li, Ruixi Yuan, and Xiaohong Guan. Accurate classification of the internet traffic based on the svm method. In *Communications, 2007. ICC '07. IEEE International Conference*, pages 1373 –1378, june 2007.

[19] M.-Y. Liao, M.-Y. Luo, C.-S. Yang, C.-H. Chen, P.-C. Wu, and Y.-C. Chen. Design and evaluation of deep packet enspection system: A case study. *Networks, IET*, 1(1):2 –9, march 2012.

[20] Annie De Montigny-Leboeuf. Flow attributes for use in traffic characterization. Technical Report CRC-TN-2005-003, Communications Research Centre Canada, Ottawa, ON K2H 8S2, December 2005.

[21] Annie De Montigny-Leboeuf and Tim Symchych. Network traffic flow analysis. In *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on*, pages 639 –642, May 2006.

[22] Open Packet. Capture repo. https://www.openpacket.org/capture/list, 2012.

[23] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[24] Randy Smith, Cristian Estan, Somesh Jha, and Shijin Kong. Deflating the big bang: Fast and scalable deep packet inspection with extended finite automata. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 207–218, New York, NY, USA, 2008. ACM.

[25] Sourcefire. Snort. http://www.snort.org/, 2012.

[26] Tcpreplay. Tcpreplay. http://tcpreplay.synfin.net/wiki, 2010.

[27] Charles V. Wright, Fabian Monrose, and Gerald M. Masson. On inferring application protocol behaviors in encrypted network traffic. *J. Mach. Learn. Res.*, 7:2745–2769, December 2006.

[28] Fang Yu, Zhifeng Chen, Yanlei Diao, T.V. Lakshman, and R.H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *Architecture for Networking and Communications systems, 2006. ANCS 2006. ACM/IEEE Symposium on*, pages 93 –102, dec. 2006.

[29] S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on Machine Learning*, pages 250 –257, nov. 2005.

[30] Fan Zhang, Wenbo He, Xue Liu, and Patrick G. Bridges. Inferring users' online activities through traffic analysis. In *Proceedings of the fourth ACM conference on wireless network security*, WiSec '11, pages 59–70, New York, NY, USA, 2011. ACM.