Peer to Peer Botnet Detection Based on Flow Intervals and Fast Flux Network Capture

by

David Zhao
B. Eng., University of Victoria, 2010

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

M.A.Sc.

in the Faculty of Engineering

© David Zhao, 2012
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.

Supervisory Committee

Peer to Peer Botnet Detection Based on Flow Intervals and Fast Flux Network Capture

by

David Zhao
B. Eng., University of Victoria, 2010

Supervisory Committee

Dr. Issa Traore, Department of Electrical and Computer Engineering
Supervisor

Dr. Kin Li, Department of Electrical and Computer Engineering
Departmental Member

Abstract

Supervisory Committee
Dr. Issa Traore, Department of Electrical and Computer Engineering
Supervisor
Dr. Kin Li, Department of Electrical and Computer Engineering

Botnets are becoming the predominant threat on the Internet today and is the primary vector for carrying out attacks against organizations and individuals. Botnets have been used in a variety of cybercrime, from click-fraud to DDOS attacks to the generation of spam. In this thesis we propose an approach to detect botnet activity using two different strategies both based on machine learning techniques. In one, we examine the network flow based metrics of potential botnet traffic and show that we are able to detect botnets with only data from a small time interval of operation. For our second technique, we use a similar strategy to identify botnets based on their potential fast flux behavior. For both techniques, we show experimentally that the presence of botnets may be detected with a high accuracy and identify their potential limitations.

Table of Contents

List of Tables

List of Figures

Acknowledgments

The writing of this thesis was as challenging as it was rewarding and would not have been possible without the continued and tireless support of my supervisor, Dr. Issa Traore. His guidance and experience were invaluable in helping me complete this work.

I am greatly indebted to my colleagues Bassam Sayed, and Sherif Saad who supported me throughout my project and assisted me tremendously when the work was at its hardest.

I would like to thank the many others who contributed their efforts to this work and offered their support and experience along the way, including Dr. Ali Ghorbani, Wei Lu, and Erik Johnson.

Most important of all I would like to thank my parents, Jenny Wen and James Zhao, who has supported me in more ways than I can possibly recount. I love you, mom and dad.

Chapter 1

Introduction

1.1 Peer to Peer and Distributed Botnets

A bot is an autonomously operating software agent which may be controlled by a remote operator (the botmaster) to perform malicious tasks typically installed onto a victim's computer without the owner's consent or knowledge. Bots allow a remote operator to control the system and command it to perform specific, typically malicious tasks. Some of the tasks performed by a botnet include distributed denial of service (DDOS), mass spam, click fraud, as well as password cracking via distributed computing and other forms of cybercrime.

Command and control (C&C) is the key identifying characteristic of a botnet, and as such there is a variety of methods used by bots to form this network structure. Command and control channels must allow a botmaster to issue orders to individual bots in an efficient manner while at the same time avoiding being detected by computer security measures. Additionally, command and control channels would ideally want to be decentralized so that individual members are difficult to detect even if the C&C channel is compromised, allowing for resiliency in the network.

One of the most popular methods for implementing botnet command and control is by using the Internet Relay Chat (IRC) protocol [1]. IRC based C&C channels are highly efficient due to the ease of implementation as well as the capability of forming large networks very quickly due to the simplicity of the network architecture. Their weakness lies in their highly centralized nature: a compromise of a botnet C&C server may compromise the location of all bots connected to that server. Additionally, monitoring of network traffic may easily reveal the messages being passed from the server to individual clients, and much research has been done on botnet detection based on the analysis of these message contents.

C&C schemes based on HTTP traffic is another popular method for botnets. As a well-known protocol, HTTP based botnet C&C attempts to be stealthy by hijacking a legitimate communications channel in order to bypass traditional firewall based security and packets are often encrypted to avoid detection based on deep packet analysis. However, HTTP based C&C schemes still suffer from the issue of centralization, and it is possible to exploit such centralized behavior in their detection.

A more recent development in botnet C&C technology utilizes peer to peer (p2p) networks and protocols to form the communications network for bots. In p2p schemes, individual bots act as both client and server, producing a network architecture without a centralized point which may be incapacitated. The network is resilient in that when nodes are taken offline, these gaps may be closed automatically, allowing for the network to continue to operate under the attacker's control. [2] [3].

Feily et al. [4] divides the life-cycle of a botnet into five distinct phases: initial infection, secondary injection, connection, malicious command and control, and update and maintenance. In the initial infection phase, an attacker exploits a known vulnerability for a target system and infects the victim machine, granting additional capabilities to the attacker on the target system. In the secondary injection phase, the attacker uses his newly acquired access to execute additional scripts or programs which then fetch a malicious binary from a known location. Once the binary has been installed, the victim computer executes the malicious code and becomes a bot. In the connection phase, the bot attempts to establish a connection to the command and control server through a variety of methods, joining the botnet officially once this connection has been established. The maintenance phase is the last phase of the botnet lifecycle, bots are commanded to update their binaries, typically to defend against new attacks or to improve their functionality.

Many existing botnet detection techniques rely on detecting bot activity during the attack phase or during the initial infection / secondary injection phase. Typical detectors are based on traditional intrusion detection techniques, focusing on identifying botnets based on existing signatures of attacks by examining the behavior of underlying malicious activities.

1.2 Fast Flux Networks

Recently, attempts have been made by researchers to shutdown or disable botnets by removing or disabling C&C servers or individual nodes from the malicious network. Such shutdown attempts operate on the concept of denying *availability* to the malicious network. If servers associated with the malicious domains may be disabled or compromised, the reduction in availability could stop or severely restrict the potential for a botnet to inflict harm.

In response to these new anti-botnet tactics, botnets have responded by employing fast flux service networks (FFSNs). The goal of fast-flux is to improve availability through a strategy of associating a fully qualified domain name with hundreds or even thousands of individual IP addresses [5]. The domain swaps these IP addresses in extremely short time intervals through round robin or similar scheduling combined with a very short Time To Live (TTL) on the DNS A record (a record of an IPv4 address which maps to the host name). In addition, the botmaster may employ load balancing mechanisms and other health checks to ensure that only healthy and available nodes are presented by the domain. Fast flux networks typically create a level of indirection for security and to increase stealth. The front end nodes are typically only redirectors to backend servers which actually serves requests. When some query is made to a malicious domain, the redirectors forward the request to the fast flux ‘mothership’ which then returns the actual response.

There are two primary categories for fast flux networks. ‘Single Flux’ networks are the more basic form of fast flux. In the single flux case, the DNS record for a malicious domain may rotate its front end node’s IP address as often as once every three minutes. In this way, even if a redirector is shut down, others standing by can quickly be fluxed into the DNS record to maintain availability.

Double flux networks provide additional redundancy by not only fluxing the DNS A records, but also the authoritative name server (NS) records. In the double flux scheme, the authoritative name server for a fast flux domain is itself part of the fast flux network.

One challenge in detecting fast flux networks comes from the difficulty in distinguishing between malicious fast flux and benign fast flux networks. Benign domains may exhibit fast flux characteristics due to the use of content distribution networks (CDNs) which employ it as a technique to improve service availability for high traffic websites.

1.3 Concept Outline

This thesis is divided into two major parts, with the first primarily focusing on the detection of botnets by analysing their network flow behaviour via a machine learning approach. In the second part, we discuss an extension of the machine learning technique which may be used to detect Fast Flux behaviour on malicious domains. We also monitor a Fast Flux network to create a geographical map of a botnet in order to gain an overview of its extent.

For the primary work, we propose a method to detect the presence of peer to peer botnets not only during the attack phase, but also in the command and control phase. We examine the network behavior of a botnet at the level of the TCP/UDP flow, splitting it into multiple time windows and extracting from them a set of attributes which are then used to classify malicious (botnet) or non-malicious traffic using machine learning classification techniques. In particular,

we compare the performance of the Bayesian Network classifier and a decision tree classifier using reduced error pruning (REPTree).

There are several advantages to detecting botnets based on their network flow characteristics. As a bot must communicate with the rest of the network during all phases after secondary injection, our approach may be used to detect the presence of a bot during a significant part of its life. Furthermore, detection based on network traffic characteristics is immune to encryption algorithms which may counter other approaches such as packet inspection and is computationally cheaper than those techniques. Additionally, by splitting individual flows into characteristic time windows, we may be able to detect bot activity quickly, before it has finished its tasks during the C&C or attack phases.

In the second part, we take the machine learning approach outlined in part one and generate a set of metrics for measuring fast flux networks and for determining whether a domain is a malicious fast flux network or part of a benign network. Additionally, by polling a fast flux domain continuously over a period of a week, we generate a geographical distribution of its front facing nodes in order to gain a picture of the physical extent of the system.

1.4 Contribution

Our work provides three major contributions towards botnet and fast flux network detection techniques.

First, our system uses flow level analysis of network traffic to detect botnets, a relatively recent approach. With flow analysis, we bypass the traditional issues of packet inspection based techniques with regards to encryption and other content obfuscating techniques used by modern botnets.

Second, our system performs detection not on the entirety of a network flow, but by analyzing botnet traffic based on small time slices. Traditional network flow detection techniques may require a significant period of time to examine the full behavior of a flow, with our system, successful classification may occur in 180 seconds once the appropriate time interval characteristics have been observed. These contributions were published in the 27th IFIP TC 11 International Information Security and Privacy Conference (SEC 2012) [33].

Third, we introduce a new approach to detect Fast Flux networks using machine learning. Many legitimate websites today use Fast Flux techniques for load balancing, and our approach can distinguish between malicious and non-malicious networks with high accuracy. This contribution was published in the Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2012) [34].

1.5 Thesis Outline

The remainder of this thesis is structured as follows:

- In Chapter 2, we give a summary of related works for botnet detection, with a focus on those techniques which use the concept of flow intervals.
- In Chapter 3, we discuss our primary detection approach, including its motivations and design.
- In Chapter 4, we evaluate our approach using experimental data and discuss the results of our evaluation.
- In Chapter 5, we build upon the techniques of the previous chapters to produce a framework for detecting fast flux behaviour for domains.
- In Chapter 6, we evaluate our fast flux detector and map botnet data we have obtained by exploiting the exposed nature of the network.

- In Chapter 7, we present the design and implementation of a prototype implementation of our botnet detector.
- In Chapter 8, we conclude by summarizing the results of our research and discuss future work that may be conducted to mitigate potential weaknesses in our system.

Chapter 2

Related Work

In this section, we examine existing research in botnet detection and analyze the strength and deficiencies of approaches based on packet inspection as well as signal analysis based on flows. We also review literature previously written in the field of Fast Flux network detection.

2.1 Flow analysis

A large collection of literature exists for the detection of botnets though interest towards the detection of peer to peer botnets has only recently emerged. Furthermore, botnet detection approaches using flow analysis techniques have only emerged in the last few years [6] and of these most examine flows in their entirety instead of smaller time intervals. Faily et al. classify botnet detection systems into four general types, signature-based detection, anomaly-based detection, DNS-based detection and mining-based detection [4]. Our focus will be on mining and anomaly based detection due to their increasing popularity.

Gu et al. proposed successively two botnet detection frameworks named BotHunter [7] and BotMiner [8].

BotHunter [7] is a system for botnet detection which correlates alarms from the Snort intrusion detection system with bot activities. Specifically, BotHunter exploits the fact that all bots share a common set of underlying actions as part of their lifecycle: scanning, infection, binary download, C&C and outbound scanning. BotHunter monitors a network and captures activity related to port scanning, outbound scanning and performs some payload analysis and malware activity detection based on Snort rules, and then uses a correlation engine to generate a score for the probability that a bot has infected the network. Like many behavior correlation techniques, BotHunter works best when a bot has gone through all phases of its lifecycle, from initial exploit to outbound scan.

BotHunter is also vulnerable to encrypted command and control channels that cannot be detected using payload analysis.

BotMiner [8] relies on the group behavior of individual bots within a botnet for its detection. It exploits the underlying uniformity of behavior of botnets and detects them by attempting to observe and cluster similar behavior being performed simultaneously on multiple machines on a network. BotMiner performs 'C-plane' clustering to first group network traffic behaviors which share similarities. Flows with known safe signatures (such as for some popular protocols) are filtered out of their list to improve performance. Once similar flows have been identified, BotMiner uses a second 'A-Plane' clustering technique which groups flows by the type of activities they represent using anomaly detection via Snort. By examining both the A-Plane and C-Plane, BotMiner correlates hosts which exhibit both similar network characteristics as well as malicious activity and in doing so identify the presence of a botnet as well as members of the network. Experimentally, BotMiner was able to achieve detection accuracies of 99% on several popular bot variants with a false positive rate around 1%.

Yu et al. proposed a data mining based approach for botnet detection based on the incremental discrete Fourier transform, achieving detection rates of 99% with a false positive rate of 14% [9]. In their work, the authors capture network flows and convert these flows into a feature stream consisting of attributes such as duration of flow, packets exchanged etc. The authors then group these feature streams using a clustering approach and use the discrete Fourier transform to improve performance by reducing the size of the problem via computing the Euclidean distance of the first few coefficients of the transform. By observing that individual bots within the same botnet tend to exhibit similar flow patterns, pairs of flows with high similarities and

corresponding hosts may then be flagged as suspicious, and a traditional rule based detection technique may be used to test the validity of the suspicion.

Zeidanloo et al. proposed a botnet detection approach based on the monitoring of network traffic characteristics in a similar way to BotMiner. In their work, a three stages process of filtering, malicious activity detection and traffic monitoring is used to group bots by their group behavior. The approach divides the concept of flows into time periods of six hours and clusters these flow intervals with known malicious activity. The effects of different flow interval durations were not presented, and the accuracy of the approach is unknown.

All of the above mentioned group behavior clustering approaches requires that malicious activity be performed by the bots before detection may occur and therefore are unsuitable for early detection during the C&C phase of a bot's lifecycle. Additionally, similarity and group behavior detection strategies relies on the presence of multiple bots within the monitored network and are unreliable or non-functional if only a single infected machine is present on the network.

Livadas et al. proposed a flow based detection approach for the detection of the C&C traffic of IRC-based botnets, using several classifiers to group flow behavior [10]. Their approach generates a set of attributes from IRC botnet flows and classifies these flows using a variety of machine learning techniques. Using a Bayesian network classifier, they achieved a false negative rate between 10% to 20% and a false positive rate of 30% to 40% though their results may have been negatively affected by poor labeling criterion of data. They showed using their approach that there exists a difference between botnet IRC chat behavior and normal IRC chat behavior and that it was possible to use a classifier to separate flows into these categories.

Wang et al. presented a detection approach of peer to peer based botnets (specifically the peer to peer Storm variants using the Kademlia based protocol) by observing the stability of control flows in initial time intervals of 10 minutes [11]. They developed an algorithm which measures the stability of flows and exploits the property that bots exhibit similar behavior in their command search and perform these tasks independently of each other and frequently. This differs from the usage of the protocol by a normal user which may fluctuate greatly with user behavior. They show that by varying parameters in their algorithm, they were able to classify 98% of Storm C&C data as 'stable', though a large percentage of non-malicious peer to peer traffic were also classified as such (with a false positive rate of 30%). Our own approach is similar to this research, though we seek to significantly increase our detection accuracy by introducing new attributes and by utilizing a machine learning algorithm.

2.2 Fast Flux

Fast flux techniques have been used by benign networks for many years as a way of distributing load for popular websites. Content management networks such as Akamai use techniques such as short TTL on DNS records, rotating IPs that are characteristics of fast flux behaviour.

The usage of fast flux techniques on malicious networks have only become popular in the last few years, and have only come to wide attention since 2007 when the honeynet project released a report of their behaviour [5] [12]. Since then, some researchers have explored ways to detect such networks and distinguish them from non-malicious fast flux networks.

Nazario and Holz presented a way of identifying and qualifying a domain as fast flux using a series of rule based heuristics. They characterized malicious fast flux networks based on a series of metrics including TTL, the existence of more than 5 unique IP's in a single A record query, and the average distance between IP addresses within the queries. They marked a domain as

'fluxy' when it exhibited at least four of the nine metrics. While they did not directly measure the accuracy of their technique, they did discover some 900 qualified domains that qualified as fast flux using the system.

Caglayan and others developed a detection technique based on measuring the TTL and two other metrics known as the fast flux activity index and footprint index. The fast flux activity index measures the degree of DNS change for a particular domain, while the footprint index measures the geographical dispersion of IP addresses associated with a single domain [13]. These data points are fed into a Bayesian classifier to determine with high accuracy the probability that a domain is malicious fast flux.

Passerini et al. used a similar approach in their fast flux detection system called FluXor. They use the age of the domain, the number of distinct DNS A records, the time to live, and other metrics to distinguish between malicious fast flux networks with non-malicious networks. Ultimately, a naïve Bayesian network detector is used to make the final detection, resulting in a detection accuracy of over 90% on fast flux networks.

Our work builds upon this work as well as our work with the detection of botnets via flow metrics to create a decision tree based approach in detection fast flux networks quickly as they exhibit fluxing behavior.

2.3 Summary

In this chapter, we reviewed past and related work in the field of flow based botnet detection as well as fast flux network detection. We examined their potential limitations and strengths towards the building of our own algorithm.

Chapter 3

P2P Botnet Detection Model

Early works in botnet detection are predominantly based on payload analysis methods which inspect the contents of TCP and UDP packets for malicious signatures. Payload inspection typically demonstrates very high identification accuracy when compared with other approaches but suffer from several limitations that are increasingly reducing its usefulness. Payload inspection techniques are typically resource intensive operations that require the parsing of large amounts of packet data and are generally slow. Additionally, new bots frequently utilize encryption and other methods to obfuscate their communication and defeat packet inspection techniques. Furthermore, the violation of privacy is also a concern in the payload analysis-based detection scheme.

A more recent technique, traffic analysis, seeks to alleviate some of the problems with payload inspection. Traffic analysis exploits the idea that bots within a botnet typically demonstrate uniformity of traffic behavior, present unique communications behavior, and that these behaviors may be characterized and classified using a set of attributes which distinguishes them from non-malicious traffic and techniques. Traffic analysis does not depend on the content of the packets and is therefore unaffected by encryption and there exists dedicated hardware which may extract this information with high performance without significantly impacting the network.

Typical traffic analysis based detection systems examine network traffic between two hosts in its entirety. While this approach is feasible for offline detection, it is not useful for the detection of botnet behavior in real time. A network flow between two hosts may run for a few seconds to several days, and it is desirable to discover botnet activity as soon as possible.

In this chapter, we present a detection technique based on traffic analysis which allows us to identify botnet activity in real time by examining the characteristics of these flows in small time windows. We exploit some properties of botnet traffic in order to perform this detection with high confidence even when other non-malicious traffic is present on the network.

3.1 General Approach

The uniformity of botnet communications and botnet behavior is well known and has been exploited by various architectures towards their detection [8] [9] [14] [15] [16]. Most of these techniques exploit this uniformity by monitoring the traffic behavior of a number of machines, and then identifying machines which are part of a botnet when they begin to simultaneously perform similar malicious actions. Other methods include observing the command and response characteristics of bots; in the BotSniffer architecture, Gu et al. detect individual bots by drawing a spatial-temporal correlation in the responses of bots to a specific command [15]. With this idea, we make the assumption that should there exist a unique signature for the flow behavior of a single bot, we can use this unique signature to detect many bots which are part of the same botnet.

Several studies have shown that it is possible to detect certain classes of network traffic simply by observing their traffic patterns. Jun et al. proposed a technique for detecting peer to peer traffic based on a set of network flow attributes [17]. While the research does not focus on computer security but instead traffic classification, they nevertheless show that it is possible to detect various classes of peer to peer applications (e.g. eMule, Kazaa, Gnutella) based on their unique flow attributes. We also observe that bots utilizing implementations of the Overnet / Kademlia p2p protocol as well as unique p2p implementations as those seen on the Waledac bot

exhibit unique and specific message exchange characteristics, particularly when first joining their p2p networks [2] [18].

For our technique, we will analyze specifically the network flow characteristics of traffic on a network. For the purposes of our framework, we define a flow as a collection of packets being exchanged between two unique IP addresses using a pair of ports and utilizing one of several Layer 4 protocols. We observe the characteristics of a given flow by examining its traffic in a given time window T and make two observations about the size of the time window. First, if a time window is too small, we may fail to capture unique traffic characteristics that only become apparent over a longer period of time, and we may also introduce errors as the behavior of a flow may change over time. If a time window is too large, we cannot make a decision in our classification until the window has been met, which means that our time to detection will increase to an undesirably long period. Ultimately, the selection of a time window size will be based on a compromise between detection accuracy and speed.

In order to classify the flow characteristics, we compute a set of attributes for each time window which encodes relevant information about the behaviour of the flow during that time window. The selection of our set of attributes is based on the observations we have made above, combined with our intuition of botnet messaging activities.

Operation of our detection framework consists of two phases. In the training phase, we provide our detectors with a set of known malicious and non-malicious data attribute vectors in order to train our classifiers in the identification of the two classes of data. Once complete, the system is placed in the detection phase, where it actively observes the network traffic and classifies the attribute vectors generated from active flows. When a set of attribute vectors has been classified as ‘malicious’ in the live data, the flows in question may be flagged as suspicious.

3.2 Attribute Selection

An attribute is some characteristic of a flow or a collection of flows in a given time window T which may be represented as a numeric or nominal value. Table 3.1 lists the set of 12 attributes we have selected for the purposes of our evaluation. Some attributes, such as the source and destination IP addresses and ports of a flow, may be extracted directly from the TCP / UDP headers, while others, such as the average length of packets exchanged in the time interval, require additional processing and computation. These attributes are then used as part of an attribute vector which captures the characteristics of a single flow for a given time interval.

We selected our set of attributes based on the behavior of various well known protocols as well as the behavior of known botnets such as Storm, Nugache and Waledac. For example, we note that unlike normal peer to peer usage, bot communication may exhibit a more uniform behavior whereupon the bot queries for updates or instructions on the network continuously, resulting in many uniform sized, small packets which continuously occur. Another observation we may make is that for many protocols, the initial exchange of packets when a client joins a network tends to be unique and follows well defined behavior; this knowledge may allow us to assist in classification by capturing the characteristics of the initial packet exchange and carrying this information forward to subsequent time intervals for that flow. For instance, the first packet size attribute is obtained immediately when the initial flow has been established and is carried on to future time windows to assist in classification.

Table 3.1: Selected network flow attributes

Attribute	Description
SrcIp	Flow source IP address
SrcPort	Flow source port address
DstIp	Flow destination IP address
DstPort	Flow destination port address
Protocol	Transport layer protocol or 'mixed'
APL	Average payload packet length for time interval.
PV	Variance of payload packet length for time interval.
PX	Number of packets exchanged for time interval.
PPS	Number of packets exchanged per second in time interval T
FPS	The size of the first packet in the flow.
TBP	The average time between packets in time interval.
NR	The number of reconnects for a flow
FPH	Number of flows from this address over the total number of flows generated per hour.

It should be noted that while included in our attribute list, the source and destination IP and port numbers for a flow may not be a very good attribute if the training data comes from a different network and uses different IP values. Typically we would like to use attributes which are universal to any network in order to provide for a more portable signature.

One final consideration for the selection of attributes is to provide some resistance to potential evasion techniques for bots. While no known bots today exhibit this evasion strategy, it is feasible that flow perturbation techniques could be used by a bot in an attempt to evade our analysis. A bot may, for example, inject random packets into its C&C communications in order to throw off correlations based on packet size. In order to mitigate some of these techniques, we measure the number of flows generated by a single address, and compare it with the number of total flows generated in some time period (in this case, an hour). This metric allows us to exploit

the fact that most bots will generate more flows than typical non-malicious applications as it queries its C&C channels for tasks and carry out those tasks. We also measure the number of connections and reconnections a flow has made over time in case the bot attempts to randomly connect and disconnect to defeat our connection based metric. Like any service, it is desirable for a bot to be connected to its command and control channel as much as possible, and therefore any random disconnects a bot performs in order to defeat detection will naturally provide some mitigation against the bot's activities. Finally, it is possible to generate white lists of known IP addresses and services which help eliminate potential benign programs which may exhibit similar connection behavior to better isolate malicious applications. None of our proposed strategies are foolproof, but they serve to increase implementation complexity for the botmaster as well as provide natural detriments to the efficient operation of a botnet.

3.3 Learning Techniques

Many traditional techniques for detecting botnets and malicious network activity consist of rule based or signature based detection. Such techniques typically utilize existing network intrusion detection systems (IDS) to correlate the presence of specific activity (such as sending spam, scanning a port, or even performing a known exploit) with the existence of a malicious agent on a given system or network. The generation of rules for such systems are often manual and performed on a case by case basis as new malware and malicious techniques are discovered. As malicious software proliferates, such manually generated rules are becoming increasingly impractical, and an automated alternative is desirable.

Machine learning techniques, similar to those used in automated spam filtering, allow detection systems to adapt quickly to rapidly changing malware and in addition allow for an automated method for discovering unique patterns and properties that may be exhibited by

malware. Our work primarily utilizes two popular machine learning approaches, the Bayesian network classifier, and the automated decision tree classifier. For purposes of our work, we would like to select classification techniques with a high performance in order to support real time detection goals while at the same time exhibiting high detection accuracy.

We have selected two popular techniques for our evaluation based on the above criteria, a Bayesian network classifier, and a decision tree classifier.

3.3.1 Bayesian networks

Bayesian belief networks belong to a family of algorithms known as the probabilistic graphical models, and are a way of representing knowledge about an uncertain domain. The nodes in the graph represent some random variable *x* while the edges between nodes represent the probabilistic dependencies between variables. In a Bayesian network, each node is a Bayesian random variable, in that they are either observable quantities, latent variables, unknown parameters or hypotheses. The Bayesian network is a directed acyclic graph: edges of the graph represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other.

Mathematically, each node of a Bayesian network is associated with a variable X_i and each edge in the graph indicates the collection of parents of any variable. Given X_i, denote the parents by $pa(X_i)$. For each node, we compute the probability of the variable X_i conditional on its parents $pa(X_i)$ and denote it by $p(X_i \,|pa(X_i))$. The graphical and probabilistic structure of a Bayesian network represents a single joint probability distribution given by:

$$P(X_1 \dots X_n) = \prod_{i=1}^{n} p(X_i \,|pa(X_i))$$

We may use the Bayesian network to classify a variable y = X_0 which we call the *class variable* given a set of variables $x = X_1 \dots X_n$called the *attribute variables*. For the purposes of our system, the class variable consists of the type of flow data, either malicious or non-malicious, and the attribute variables are represented by the distinct flow attribute values that we compute from the flow information. A classifier *h:* $x \rightarrow y$ is a function which maps an instance of *x* to a value of *y*. We generate this classifier from a dataset *D* consisting of samples over (*x*, *y*). Once we have constructed such a network, we calculate the $argmax_y$p(y|x) using $p(X_1 \dots X_n)$ and note:

$$P(y|x) = \frac{P(X_1 \dots X_n)}{P(x)}$$

$$= \prod_{i=1}^{n} p(X_i \,|pa(X_i))$$

As we know all variables in the set *x*, we simply compute the probability distribution for all class values in order to classify *y*.

Several algorithms exist for computing the Bayesian network which classifies some dataset *D*. Because a Bayesian network is a directed graph with probabilities, it is natural for algorithms to generate such a network using a two-step process, with the first step being to learn the *structure* of the network and a second step to learn the individual probabilities. Our system uses the Weka machine learning library to construct the Bayesian network for our dataset. Weka uses a search algorithm to determine the structure of the network, and then uses an estimator algorithm to generate the probabilities for the network. For search, a hill climbing algorithm is used to generate a non-optimal graph and then discover a local optimum. For the estimator, we utilize a simple estimator which provides a direct estimate of the conditional probabilities:

$$P(x_i = k|pa(x_i) = j) = \frac{N_{ijk} + N'_{ijk}}{N_{ij} + N'_{ij}}$$

where N'_{ijk} is an 'alpha' parameter where alpha = 0 obtains the maximum likelihood estimates. Figure 3.1 and Figure 3.2 illustrates the logical construction of the Bayesian Network. In Figure 3.1, the layout of the network can be seen, with each relationship between type and the metric being represented by a probability table which gives the probability of a certain type occurring for a given value of the metric. Figure 3.2 shows a small subset of this table for the possible values of the metric 'PV'. Using these probabilities we can infer questions such as: given the value of PV and other variables, what is the likelihood of 'Type' being malicious?

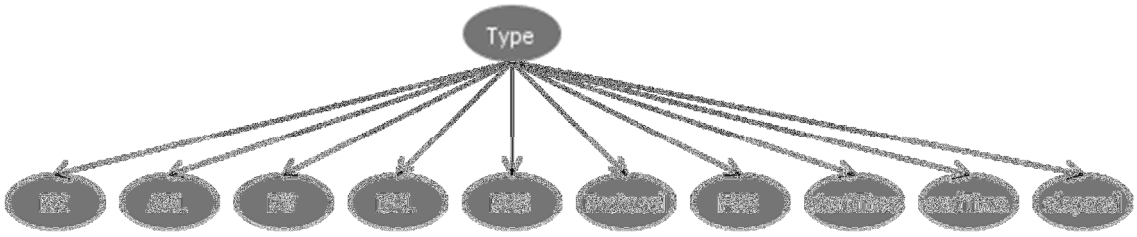

Figure 3.1: An example belief network based on sample data; each node has a probability table representing the conditional probabilities

Type	'(-inf-1.5]'	'(1.5-2.5]'	'(2.5-8.5]'	'(8.5-13.5]'	'(13.5-14.5]'
nonmalicious	0.464	0.001	0.013	0.007	0.001
malicious	0.014	0.049	0.009	0.036	0.088

Figure 3.2: A small subset of the probability table for the packet variance variable. The top row indicates the value of the packet variance and the bottom rows indicate the probability of a flow being non-malicious or malicious given those values.

3.4.2 Decision trees

Automated decision tree classifiers uses a decision tree as a predictive model, taking in a series of observations and generating a value for those given inputs. Leaf nodes in a decision tree represent a class or a node linked to two or more subtrees called a *test* node. At a test node, some

outcome is computed based on the attribute values of an instance, where each possible outcome leads to one of the subtrees rooted at the node.

In a decision tree, interior nodes represent input attributes with edges extending from them which correspond to possible values of the attributes. These edges eventually lead to a leaf node which represents an output variable (in our case, whether a flow is malicious or non-malicious). Classification of an attribute vector simply consists of traversing the path from root to leaf by following edges which correspond to the appropriate values of the attributes in the attribute vector. Decision trees are learned via a partitioning process where the source attribute set is split into subsets based on a value test. This partitioning halts after a user-defined stopping criteria has been reached.

Decision trees, due to their structural composition, are easy to visualize and interpret by a human and produce a white box model which allows for easy analysis of the generated classifier. Automated algorithms which construct decision trees are prone to over-fitting data, creating many branches in the tree in order to capture very specific and complex scenarios in a given training set. Pruning algorithms are therefore used on decision trees in order to trim the size of the tree and reduce its complexity.

Two major pruning methods are *post-pruning* and *online pruning*, and they differ mainly on the timing of their implementation. For any pruning algorithm, the goal is to determine the amount of error that a decision tree would suffer before and after each pruning stage, and then decide how to prune to avoid error. The estimate of the possible error is determined using the *error estimate* given by:

$$E = \frac{e + 1}{N + m}$$

where *e* represents misclassified examples at a given node, *N* represents the number of examples that would reach the node, and *m* represents the set of all training examples.

Post-pruning strategies are performed on fully induced decision trees. By sifting through the tree from the bottom up, the probability of sibling leaf nodes are compared and nodes which show overwhelming dominance will be pruned.

Online pruning methods prune decision trees while it's being induced. During decision tree creation, an algorithm divides the data set on attributes that provide the most information gain about a given class; this dividing factor is used as the 'deciding' attribute. When a split is made, if a child leaf that represents less than a minimum number of examples from the dataset, the parent node and all its children are reduced to a single node. Online pruning techniques are good at reducing the size of a decision tree, but carry a heavier penalty with respect to accuracy.

For our evaluation, we select a decision tree using the Reduced Error Pruning algorithm (REPTree). This algorithm helps improve the detection accuracy of a decision tree with respect to noisy data, and reduces the size of the tree to decrease the complexity of classification.

The resulting trees may be seen in Figure 3.3 and Figure 3.4. In 3.3, a single branch of the tree is given in graphical form, while in 3.4 a portion of the tree's nodes are shown. In reality, the actual tree spans several thousand nodes and is difficult to represent on a single sheet of paper due to its graphical complexity.

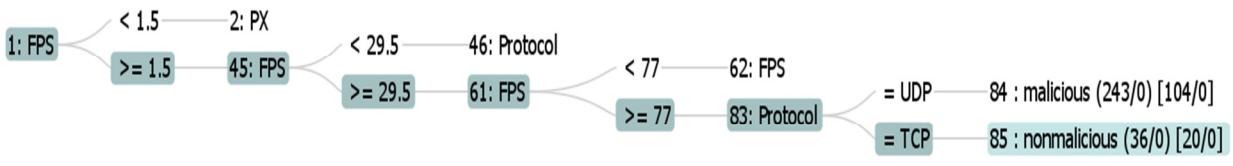

Figure 3.3: Resulting decision tree produced by our algorithm on evaluation data. Here, the highlighted branch indicates a decision path which points to a likely non-malicious traffic. Each node indicates a

decision point, for example, the root indicates that if FPS is <1.5, then the 'PX' value should be examined.

```
FPS < 1.5
|   PX < 10.5
|   |   elapsed < 2998.5 : nonmalicious (5456/4) [2695/4]
|   |   elapsed >= 2998.5
|   |   |   Protocol = UDP
|   |   |   |   elapsed < 4915 : malicious (6/0) [0/0]
|   |   |   |   elapsed >= 4915
|   |   |   |   |   elapsed < 9354.5 : nonmalicious (164/0) [89/0]
|   |   |   |   |   elapsed >= 9354.5
|   |   |   |   |   |   elapsed < 15061.5 : malicious (14/0) [6/1]
|   |   |   |   |   |   elapsed >= 15061.5
|   |   |   |   |   |   |   elapsed < 179859
|   |   |   |   |   |   |   |   elapsed < 29849.5 : nonmalicious (29/1) [12/0]
|   |   |   |   |   |   |   |   elapsed >= 29849.5
|   |   |   |   |   |   |   |   |   PX < 2.5 : nonmalicious (6/1) [0/0]
|   |   |   |   |   |   |   |   |   PX >= 2.5 : malicious (9/4) [8/3]
|   |   |   |   |   |   |   elapsed >= 179859 : nonmalicious (84/0) [31/0]
|   |   |   Protocol = TCP : nonmalicious (588/11) [282/3]
```

Figure 3.4: Textual representation of a decision tree generated on live network data. Here, the right hand side nodes indicates a path that will lead to a classification decision ending on PX <2.5 or PX >= 2.5.

3.5 Computing attributes

In order to perform classification and detection using the Bayesian network, we use an automated system to examine network traffic, aggregate the traffic into flows, and compute the flow attributes as selected in the previous section. The process of examining network traffic is done on a packet by packet basis.

The detector performs live packet capture and the packet of each header is examined to determine the source and destination IP addresses of the packet. Using this information, the packet is aggregated with other packets seen previously which belong to the same flow. The time of when the packet was received is also recorded in order to keep track of the active duration of a flow, and once the time interval *t* indicated by the detector has been reached, all packets aggregated for that flow up to the time *t* is processed.

The detector computes from the collection of packets for a flow all attributes required for classification (i.e. average packet size, first packet size, etc.) and generates a vector data structure with all attribute information. This attribute vector data is then sent to the classifier for classification and detection.

3.6 Summary

In this chapter, we discussed the general approach for our detection algorithm and gave a brief look at the types of classifiers used. We describe the process for classification as separating flows into ‘feature vectors’ which are then used to classify a flow as malicious or non-malicious. In the next chapter, we describe the experimental evaluation of our approach discuss the obtained results.

Chapter 4

P2P Detection Model Evaluation

In this chapter, we present our approach in evaluating our P2P botnet detection approach. In the first section, we introduce our test dataset and briefly describe our data generation methods. In subsequent sections, we discuss the results of our tests run on the dataset.

4.1 Dataset

There are considerable difficulties in obtaining real world datasets of botnet malicious activity. Many publicly available datasets consist of information collected from honeypots which may not reflect real-world usages. In a typical honeynet configuration, a honeypot is a machine dedicated for the collection of malicious data and typically is not used for other normal activities. In such a case, it is atypical to see non-malicious traffic within a honeypot network trace except in the smallest quantities, and such non-malicious data rarely reflect real world usage scenarios.

In order to evaluate our system, we attempt to generate a set of network traffic traces which contain both malicious and non-malicious traffic, including traffic from standard usage of popular networked applications. Malicious and non-malicious traffic are intermixed in a way that both types of traffic occur during the same time periods, and we label this data in order to evaluate the accuracy of our methods.

For this work, we obtained and used two separate datasets containing malicious traffic from the French chapter of the honeynet project involving the Storm and Waledac botnets, respectively [19].

To represent non-malicious, everyday usage traffic, we incorporated two different datasets, one from the Traffic Lab at Ericsson Research in Hungary [20] and the other from the Lawrence Berkeley National Laboratory (LBNL) [21]. The Ericsson Lab dataset contains a large number of

general traffic from a variety of applications, including HTTP web browsing behavior, World of Warcraft gaming packets, and packets from popular bittorrent clients such as Azureus.

The LBNL is a research institute with a medium-sized enterprise network. The LBNL trace data consists of five datasets labeled D_0...D_4; Table 4.1 provides general information for each of the datasets.

Table 4.1: LBNL datasets general information

	D_0	D_1	D_2	D_3	D_4
Date	4/10/04	15/12/04	16/12/04	6/1/05	7/1 /05
Duration	10min	1hour	1hour	1hour	1hour
Subnets	22	22	22	18	18
Hosts	2,531	2,102	2,088	1,561	1,558
Packets	18M	65M	28M	22M	28M

The recording of the LBNL network trace happened over three months period, from October 2004 to January 2005 covering 22 subnets. The dataset contains trace data for a variety of traffic which spans from web and email to backup and streaming media. This variety of traffic serves as a good example of day-to-day use of enterprise networks.

In order to produce an experimental dataset with both malicious and non-malicious traffic, we merged the two malicious datasets and the Erikson (non-malicious) dataset into a single individual trace file via a specific process depicted by Figure 4.1. First we mapped the IP addresses of the infected machines to two of the machines providing the background traffic. Second, we replayed all of the trace files using the TcpReplay tool on the same network interface card in order to homogenize the network behavior exhibited by all three datasets; this replayed data is then captured via wireshark for evaluation.

The final evaluation data produced by this process was further merged with all five LBNL datasets to provide one extra subnet to indeed simulate a real enterprise size network with thousands of hosts. The resulted evaluation dataset contains 22 subnets from the LBNL with

non-malicious traffic and one additional subnet as illustrated in Figure 4.1 (next page) with both malicious and non-malicious traffic originating from the same machines.

4.2 Results

We implemented our framework in Java and utilized the popular Weka machine learning framework and libraries for our classification algorithms [22]. Our program extracts from a given pcap file all flow information and then parses the flows into relevant attribute vectors for use in classification.

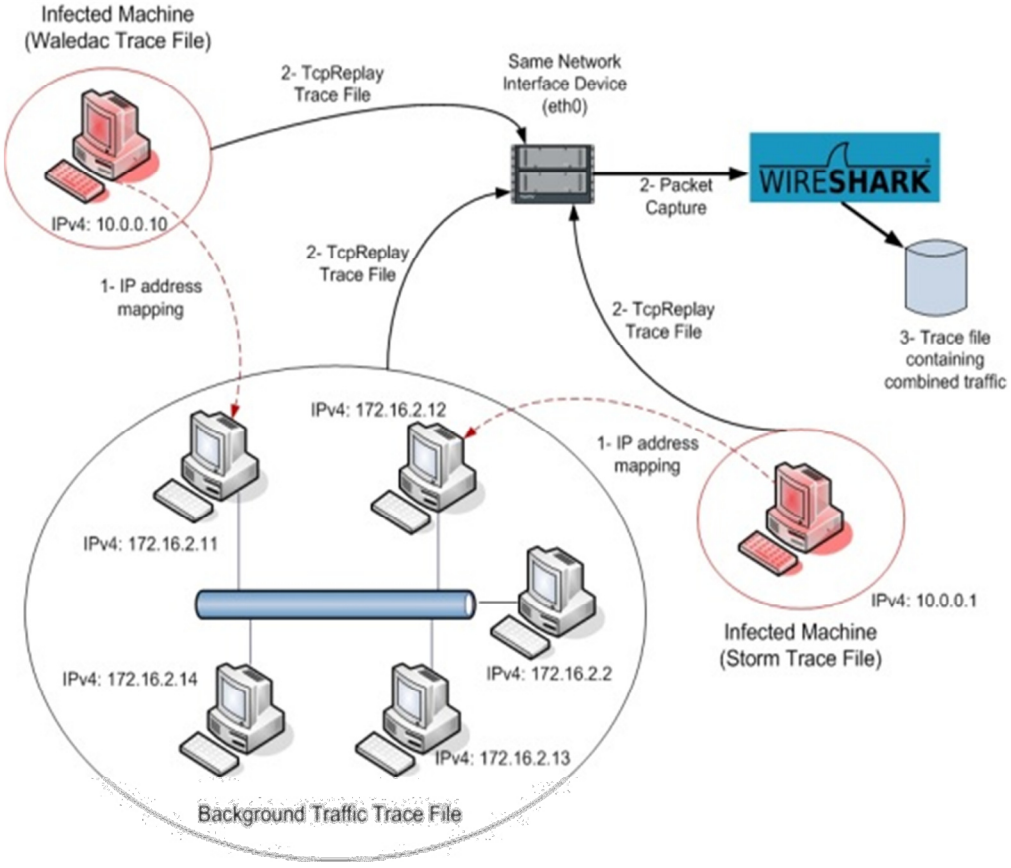

Figure 4.1 Dataset merging process

In all, there were a total of 1,672,575 network flows in our test set. The duration of the flows vary greatly, with some lasting less than a second and a few lasting more than a week. Of these

flows, 97,043 (about 5.8%) were malicious, and the remainder non-malicious. From these flows, we generated 111,818 malicious attribute vectors, and 2,203,807 non-malicious attribute vectors. Each feature vector represents a 300 second time window in which at least 1 packet was exchanged. We consider malicious flow attribute vectors a vector which is extracted from a flow associated with the Storm or Waledec botnet data, and we considered all other attribute vectors as non-malicious, including peer to peer applications such as Bittorrent, Skype and e-Donkey.

To evaluate detection accuracy, we used the 10-fold cross-validation technique to partition our dataset into 10 random subsets, of which 1 is used for evaluation and 9 others are used for training. This process is repeated until all 10 subsets have been used as the testing set exactly once, while the remaining 9 folds are used for training. This technique helps us guard against Type III errors and gives us a better idea of how our algorithm may perform in practice outside of our evaluation data. The true and false positive rates of both the Bayesian Network and decision tree classifiers are listed in Table 4.2 and 4.3 respectively. The resulting detection values are an average of the results of the ten runs.

Table 4.2: Detection rate of Bayesian Network Classifier (attribute vectors identified)

Detection rate using Bayesian Network classifier (T = 300s)		
	True positive	False positive
Malicious	97.7%	1.4%
Non-Malicious	98.6%	2.3%

Table 4.3: Detection rate of REPTree classifier (attribute vectors identified)

Detection rate using decision tree classifier (REPTree) (T = 300s)		
	True positive	False positive
Malicious	98.3%	0.01%
Non-Malicious	99.9%	1.7%

As can be seen in the above tables, both the Bayesian Network classifier and the decision tree produced very high (above 90%) detection rates with a very low false positive rate. Between the

two classifiers, the decision tree was more accurate, classifying 99% of all attribute vectors correctly while incorrectly identifying only 1% of all attribute vectors. These results indicate that there are indeed unique characteristics of the evaluation botnets when compared to everyday p2p and non-p2p traffic.

In terms of speed, the decision tree classifier was slightly slower than the Bayesian Network classifier, requiring 76.9 seconds for the full evaluation as compared to 56.1 seconds for the Bayesian Network.

In order to get some idea of the key discriminating attributes in our dataset, we use a correlation based attribute evaluator (CFS feature set evaluation) with best first search to generate a list of attributes with the highest discriminatory power while at the same time exhibiting low correlation with other attributes in the set. The algorithm generated a subset of four attributes, listed in Table 4.4, to be used as an optimized subset that may improve our performance without producing a large reduction in accuracy.

Table 4.4: Attribute subset from CFS subset evaluation

Feature	Description
PV	Variance of payload packet length for time interval.
PX	Number of packets exchanged for time interval.
FPS	The size of the first packet in the flow.
FPH	# flows per address / total flows

Tables 4.5 and 4.6 list the results of classification using only the above attribute subset. We can see that by reducing the number of attributes to three, the accuracy of the Bayesian network and REPTree classifiers both decreased slightly due to an increased false positive rate. Neither classifier with the reduced attribute set performed as well as the REPTree classifier with the full attribute set, though both very closely matched the best case's accuracy. In terms of performance, reducing the number of attributes allowed the Bayesian network classifier to

classify all attribute vectors in 76% of the original time, while the REPTree classifier took 33% of the time. Table 4.7 shows the actual times for classifying all attribute vectors by the classifiers on both the full attribute set and the reduced set.

Table 4.5: Detection rate of Bayesian Network Classifier with reduced subset

Detection rate using Bayesian Network classifier (T = 300s)		
	True positive	False positive
Malicious	92.3%	1.9%
Non-Malicious	98.1%	7.7%

Table 4.6: Detection rate of REPTree classifier with reduced subset

Detection rate using decision tree classifier (REPTree) (T =300s)		
	True positive	False positive
Malicious	98.1%	2.1%
Non-Malicious	97.9%	1.9%

Table 4.7: Classifier performance (average training time)

Performance of classifiers	
Classifier	Time (seconds)
BayesNet	40.6
REPTree	29.4
BayesNet (subset)	11.22
REPTree (subset)	8.58

With the above results, we may conclude that both the Bayesian network classifier and the decision tree classifier are suitable for the building of a botnet detection framework based on flow attributes. We further observe that while maximum accuracy may be achieved with a sizable attribute vector, we may be able to detect unique characteristics in bot traffic based simply on their packet exchange behavior, in particular the variations in their flow behavior compared to standard network traffic and the first packet exchanged in any new flows created by such malicious traffic.

Finally, we examine the effects of varying the size of our time window on the accuracy of detection. Figures 4.2 and 4.3 show the effects of the time window size on the true and false positive rates for malicious flow detection. The best results were obtained when T = 180 seconds

for both the REPTree and BayesNet classifiers, though very good results were obtained for all four time window sizes (10, 60, 180 and 300 seconds).

While both the performance and accuracy of our classifiers are satisfactory for real time detection, it must be noted that the system may be sensitive to new behaviors from bots implementing highly varied protocols. In order to guard against such a threat, a method for online training and continuous refinement of the classifiers must still be implemented.

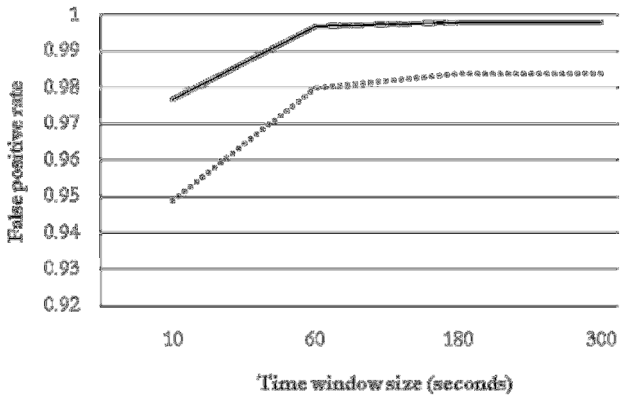

Figure 4.2: Effects of time window size on true positive rate for REPTree *(solid line)* and BayesNet *(dotted line)* classifiers.

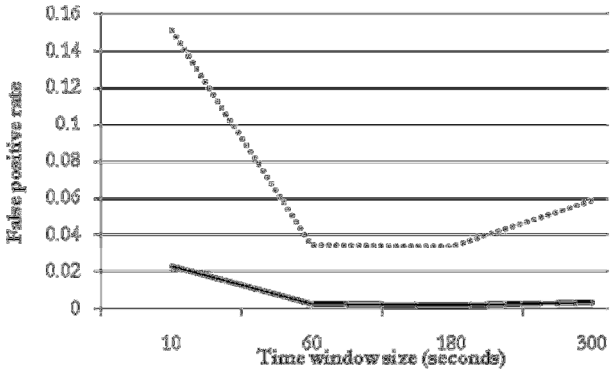

Figure 4.3: Effects of time window size on false positive rate for REPTree *(solid line)* and BayesNet *(dotted line)* classifiers.

4.3 Summary

In this chapter, we evaluated the accuracy of our algorithm by testing it against a real world dataset. After testing, we found a promising result of 98.3% true positive and 0.01% false positive rate on the detection of malicious botnet activity using our REPTree classifier, and a 97.7% true positive and 1.4% false positive rate using our Bayesian Network classifier.

Chapter 5

Fast Flux Network Detection Model

In this chapter, we present an introduction on Fast Flux Networks and the problem of Fast Flux Network classification. We propose a machine learning approach to Fast Flux Network classification that can differentiate between malicious and non-malicious fast flux networks.

5.1 Characteristics

The detection of FFSNs is motivated by the increasing prevalence of fast flux techniques within the botnet ecosystem. As malware authors explore new avenues to exploit their botnets, the concept of botnet based hosting has become an increasingly popular way for botnet authors to earn an income. The challenge facing these botnet based hosting services is similar to those of typical legitimate websites, namely, availability. Typical uses for botnet hosted sites include malware distribution, advertisement, spam and phishing attacks, and so aside from the normal challenges faced by a typical website, botnet hosted content are also frequently illegal in nature and so faces the risk of being actively disabled by enforcement agencies [23].

Malware authors therefore turn to fast flux networks, a strategy first employed by legitimate sites, in order to maintain availability in response to these pressures. Because fast flux behaviour exists for both malicious and non-malicious websites, it is therefore critical for any detection strategy to be able to distinguish between the two. Such a detector must be aware of the subtle but critical differences which distinguish the two types, as well as the inherent properties of a malicious fast flux network in order to avoid overlooking a potential malicious domain.

The Honeynet Project [5] identified two types of fast flux networks, single flux and double flux, where single flux consists of the fluxing of DNS ‘A’ records while double flux also fluxes the NS records for a domain. In either case, the average rate of change for the returned records

associated with a fast flux domain is approximately 3 to 10 minutes, and in the double flux case a series of between five to ten A records and five Name Server (NS) records are returned in a round robin fashion. In a traditional phishing example, the shutting down of a single IP address will result in the shutdown of the hosting machine and therefore the scam site itself, but in the fast flux case, the shutdown of a single IP address correlates only to the shutdown of one of thousands of potential zombies which serve content redirected from a 'mothership', and is ineffective at halting the operation. In double flux, even the DNS nameservers are in reality proxies for a command and control system which is protected behind the scenes, its content easily replicated to other IP addresses that have been infected.

All fast flux network share certain characteristics which may be used to identify them, but the existence of non-malicious networks which share these characteristics complicates the problem of detection.

Low Time-to-live

One of the defining characteristics of all fast flux networks is a low TTL value on the A records. The TTL value in DNS specifies how long a nameserver should cache a particular record before it should query the authoritative nameserver to refresh the record. If a stub resolver queries the nameserver for a record before the cache is expired, the server simply returns the cached value with no change. A low time to live is therefore important for a fast flux network to ensure that any queries made to a particular domain always quickly reaches the latest and highest available nodes. Traditional web sites tend to exhibit a very long time to live (a common value is 24 hours, or 86400 seconds), but in a fast flux network this value may be as low as 300 seconds.

Low time to live values almost always identifies a network as fast flux, but due to the increasing load on popular websites today, the introduction of CDNs and fast flux techniques

have made it a metric that will typically catch malicious and non-malicious sites alike. Some popular non-malicious websites with very low TTL can be seen in **Table 5.1**: TTL of popular domains

Table 5.1: TTL of popular domains

TTL values of popular non-malicious websites	
Domain	TTL
Google.com	300
Amazon.com	900
Microsoft.com	3600
Ebay.com	3600

Number of unique A records

While both malicious and non-malicious domains may exhibit a low TTL value, the IP addresses returned in a DNS query from a malicious domain can differ from a non-malicious domain in certain ways. One such difference is in the number of unique A records returned over time. Popular websites using content distribution networks typically have a small set of IP addresses serving some geographical area, with each IP corresponding to a powerful backend server capable of servicing many requests. On the other hand, the A records returned from malicious domains tend to be weaker, zombie machines of which there are tens of thousands. A DNS query made to a malicious fast flux domain therefore is likely to discover many more unique IP addresses over time, and this metric may be a powerful indicator of malicious activity.

IP Networks

In many non-malicious domains, the set of IP addresses returned correlates strongly with each other as they are often part of the same network clustered around a specific geographical location while malicious fast flux networks do not share this quality. In malicious fast flux, the geographic distribution of nodes tend to be more widespread and individual machines are not closely associated with each other as they tend to be personal computers on individual networks. IP address distance measures whether IP addresses returned from a DNS query are part of the

same network or subnetwork with the expectation that non-malicious networks will contain addresses belonging to a small number of distinct networks.

IP Geolocation

IP addresses in a non-malicious fast flux domain will tend to be geographically clustered to provide for the best service, this is not likely to be true for malicious fast flux domains as the geographical distribution depends on the locations of the infected machines.

Change consistency

Observations of malicious fast flux service networks in the wild indicates that they may lie 'dormant' for long periods of time in which no fast flux behaviour is exhibited. In these cases, fast flux behaviour is only observed when the network is used to perform some active malicious task, and these tasks may last a few weeks [23]. This behaviour is not typically observed in non-malicious fast flux networks which rotate its IP address set on a more consistent basis.

Domain lifetime

We may obtain additional information on a domain by querying its whois record for the lifetime of the domain name. Malicious domains typically do not last very long as they are frequently taken down by legal authorities or in other cases by the malware authors themselves as the domains become blocked on relevant anti-malware lists. On the other hand, non-malicious domains are generally much longer lasting, existing for years after creation. By comparing the time of domain registration in whois records, we can use this value as a discriminator in our detection algorithms.

5.2 Attribute Selection

The generation of attributes for our fast flux detection framework is motivated by the characteristics identified in the previous section. Because malicious fast flux networks may have

their fluxing behaviour turned on or off by the malware author depending on the situation, we cannot guarantee that monitoring a domain in a short time interval will fully capture fast flux behaviour. Because of this fact, our detector should be conscious of which domains should be subject to further monitoring, and which should be passed over for more suitable targets for monitoring.

In order to provide this behaviour, we develop our detection system with two phases. In the first phase a fast rule based detector observes several attributes of the domain and makes a decision on whether further monitoring is needed. If the characteristics of the domain do not suggest malicious fast flux behaviour, the domain is not scheduled for extended monitoring. If, on the other hand, the initial rule based detector finds signs that a domain does exhibit characteristics of malicious fast flux, it is monitored extensively over a long period of several days in order to more closely observe its behaviour.

Filtering stage

In order to determine if a domain should be monitored for an extended period, we must first discover if it is immediately obvious that the domain is fast flux or not. To do this, we first run our attribute collection framework on the domain and collect information for a period of 1 week. If our confidence of fast flux during this monitoring period is low, or if we simply did not detect any fast flux behavior, we use the domain creation date as a judge of whether the domain is worth further monitoring. Domains created a year ago or older are discarded, while domains created within the past year are flagged for further monitoring. The monitoring phase serves to eliminate obvious non-malicious domains from our detector list so we can utilize these resources to monitor more likely targets.

Data capture stage

Once a set of domains to monitor have been finalized, the detector moves onto the data capture stage. During this stage, a DNS record is polled continuously at a rate of ½ it's A record's TTL, and the responses captured. We choose ½ the A records TTL arbitrarily as a number that is smaller than the TTL in order to ensure that we do not miss the TTL refresh when polling the nameserver. The valuable datasets captured during this process include the A and NS recorded returned and their individual TTL's. The data captured through this process are then transformed into a set of attributes which we may feed into our decision tree classifier. The attributes we generate from this dataset consists of the following:

Mean Reported TTL (MTTL)

It is possible that during the polling period we witness a change in TTL value. This metric computes the average of the TTL values returned over the polling period. If the TTL does not change, MTTL = TTL. We use this attribute to capture the 'low TTL' behavior of fast flux networks.

Actual Mean TTL (ATTL)

There have been observations that in several malicious fast flux networks, the actual rate of flux is even lower than the given TTL value for a DNS record [13]. Typically, non-malicious networks respects the TTL values and do not deviate heavily from it. The ATTL attribute records the mean time of actual change of IP addresses for a given DNS record. As we are limited in resolution by our polling interval, the lowest value ATTL can achieve is half the MTTL. Difference between ATTL and MTTL may suggest malicious behavior, and a low ATTL suggests fast flux behavior.

Total Unique A Records (ARCRD)

Motivated by the number of A records characteristic, this attribute is a count of the number of unique A records witnessed during the polling period. A high ARCRD may suggest fast flux behavior, while a very high ARCRD record may suggest malicious behavior.

A Record Change Variance (ARCRDV)

This attribute captures the variance in the number of A record changes witnessed over the polling period and is motivated by the change inconsistency characteristic of malicious fast flux networks. For most non-malicious networks, we expect the variance to be low, as records will tend to change on a consistent basis, while a malicious domain may exhibit a larger variance as the operators turn the fast flux behavior on and off depending on activity.

A Record IP Stability (ARCRDS)

We capture the network differences in the IP addresses returned by measuring its 'stability'. This is done by measuring the difference in the octets for each IP address returned in a record. We expect non-malicious domains to exhibit a smaller difference per record than a malicious domain (most may share higher order bits as they belong to the same network).

Domain name confidence (DCONF)

The domain name confidence rating is computed from the age of the domain and its domain registrar. This value ranges from 0 – 1.0, where 0 specifies a low confidence in the domain and 1.0 specifies a high confidence. We use two simple characteristics to compute this value: the age of the domain name, and whether the registrar of the domain name exists on a blacklist. Blacklists of malicious domains are maintained in several large lists readily available on the internet. If the age of the domain exceeds 1 year, and the domain does not exist on our blacklist, the value is 1. Otherwise, the value is 0 or 0.5 depending on whether one or none of the criteria mentioned above is fulfilled.

5.3 Summary

In this chapter, we described the key characteristics of a fast flux network, and provided a brief summary of the distinguishing characteristics of a malicious fast flux network versus a traditional fast flux network. Finally, we provided a small list of attributes which may be computed by examining DNS / WhoIs query data as well as various backlists which may be used to classify unknown servers as malicious or non-malicious fast flux.

Chapter 6

Fast Flux Network Detection Model Evaluation

In this chapter, we present the dataset used to evaluate our detection model and present the results of our evaluation run. Finally, we give a detailed geographic map of all malicious Fast Flux Networks detected by our system during our evaluation period.

6.1 Dataset

We evaluate the effectiveness of our algorithm at detecting fast flux domains by using a list of malicious domain names provided by the DNS-BH project [24]. The project provides a constantly updated list of over 70,000 domains known to serve malware or are known to be associated with malicious activities. Additionally, we use a community created malware domain list containing 80,000 active and inactive malicious domains discovered between 2009 and 2012. This list contains domains from the popular Zeus botnet as well as Waledac bots.

A custom build application is used to monitor a random sampling of these domains over a period of 3 days to collect data and observe for possible fast flux activity. Amongst the domains in our dataset, approximately 200 have been confirmed by a third party to have exhibited fast flux behavior, for fast flux behavior observed on other domains, we manually verify the accuracy of detection.

We also include in our sample data a selection of 300 known non-malicious domains pulled from Alexa's top 500 internet sites. All sites on this list are high traffic, high demand websites and typically use a variety of techniques for load balancing, including the use of CDN's and fast flux behavior. We use these sites to check for the false positive rates of our detector and fine tune its behavior to address any false positives we may find.

6.2 Results

Malicious fast flux networks frequently exhibit transient behavior. As our detection algorithm inspects networks for fast flux behavior and then makes a decision on whether that network is malicious, we must first observe fast flux behavior before we can classify the network.

For purposes of evaluation, we cease monitoring a network after 7 days if we noticed no fast flux behavior and our confidence in the network's maliciousness is low. The algorithm is configured to examine roughly 2,000 domains simultaneously, and three machines were used, resulting in an evaluation period of roughly 2 months. Because verification of fast flux behavior must be done manually, or by label, we use only a small subset of 500 domains as a test for our algorithm's accuracy. For the remainder, we simply list the results reported by the algorithm without hand verification. **Table 6.1** gives the detection accuracy for our small sample. All 100% of domains labeled and verified as malicious fast flux exhibited fast flux behavior and were detected as malicious.

99.0% of domains labeled non-malicious were detected as fast flux networks and selected as being non-malicious, with 0.5% (1 site) that was mistakenly identified as malicious. The misidentified site (imageshack.us) presented interesting behavior that resembled those of malicious networks, namely, the domain resolved to a large number of IP addresses (over 100), and exhibited very low TTL. In future improvements to the detector, a more discriminatory domain name confidence rating may be used to eliminate such errors. In the case of imageshack.us, the age of the domain registrar and the fact that all IP's belonged to the same network could have been used to increase the confidence rating and assist in it being identified correctly.

Table 6.1: Detection rate of malicious fast flux on sample of 500 domains

Detection rate of malicious fast flux over 500 domains		
	True positive	False positive
Malicious	100.0%	0.5%
Non-Malicious	99.5%	0.0%

Table 6.2: Count of domains detected as fast flux

Fast flux domains detected in evaluation	
	Count
Malicious fast flux	10,312
Non-malicious fast flux	9870
Non fast flux or dead	69,688

Table 6.2 gives the count of fast flux and non-fast flux behavior detected by our algorithm. Of the 80,000 malicious domains tested, 10,312 were found to be fast flux networks or have exhibited some fast flux behavior. The remainder either did not exhibit any fast flux behavior over the evaluation period, or were moved / dead.

6.3 Geomapping

With the results in 6.2, we may take the resolved IP addresses of the malicious fast flux networks and obtain geolocation data on the IP's. Figure. shows this data plotted on a map of the world, with each pin indicating a malicious IP address that was found to be a part of a malicious fast flux network.

It can be observed from the geomap that malicious fast flux agents are widely distributed across the globe. For the networks discovered in our evaluation, most IP addresses originated from the United States, Europe, or China, with a few from other countries such as India and South Africa.

As we would expect, there is very little correlation between the domain of a site and the fast flux agents servicing that domain. Many domains in our list uses the top level domain name for Russia (.ru) but most flux agents servicing these domains were found outside of that country.

6.4 Summary

In this chapter, we showed that our fast flux network detection model is capable of detecting fast flux nodes with high accuracy, but acknowledge that due to the speed of detection and the sheer difficulty of verification a more comprehensive evaluation will take significantly more work.

Figure. 6.1: Map of 10,000 malicious fast flux IPs

Chapter 7

Live implantation

In this chapter, we present an overview of the web-based, real time detection system we created to utilize this algorithm in a production environment.

7.1 Algorithm overview

Our algorithm may be used for both offline detection as well as live detection in a production environment. In order to perform further evaluations as well as demonstrate the efficacy of the system, we have begun work on a web based detection system which may be deployed in a live setting. Figure 7.2 shows the system as it is viewed on a web browser.

The live version of the botnet detection system was written in Java, using Google Web Toolkit (GWT) for the presentation layer. Communication between the HTML / Javascript based user interface and the server side code is handled by the GWT Remote Procedure Call (RPC) mechanism. High performance Javascript communicates with the server via a single Java Servlet which then dispatches the request to runtime injected servlets managed by Google Guice.

On the server side, data is stored and retrieved from a database managed by an Object Relational Mapper (ORM) based on the Java Persistence API (JPA2). By using an ORM, we add an additional layer of abstraction between the server side code and the database, allowing us to utilize numerous supported relational databases. The introduction of the ORM layer for persistence as well as the usage of GWT allows our system to be implemented entirely using native Java objects, increasing maintainability and compatibility.

A series of worker threads are created as needed from a thread pool in order to perform tasks supported by the detection system. Typical tasks include generating a signature file, reading and processing packet data, and performing live detection.

The system is capable of performing both live detection as well as offline detection based on existing datasets. Assessment and refinement of the system's capabilities are still ongoing.

7.2 Architecture

The architecture of the web based detector implementation consists of a standard Model, View, Controller (MVC) pattern with additional layers that separates the persistence (database) layer and the hardware interface layer from the rest of the system. Figure 7.1 shows the high level block diagram of the application architecture.

The 'View' layer of the system is responsible for presenting a graphical user interface (GUI) to the end user which is to be viewed on a web browser. The view primarily handles two types of operations, purely 'client side' actions by the user which has no influence on the server, and making 'server side' requests that are then handled by the server in some way. Client side actions primarily deal with presenting a responsive interface to the end user, for example, the depressing of a button when it is clicked, or the sorting of a list or graphic are client side operations. The purely client side portions of the view is structured using HTML markup and styled using Cascading Style Sheet (CSS). The user interface is made highly dynamic through the use of high performance JavaScript generated by the GWT compiler. This JavaScript code actively manipulates the Domain Object Model (DOM) of the page to create effects such as popup dialog boxes, text highlighting and other changes. Server side requests are required when an user indicates that some server operation must be performed (such as uploading a file to the server, or requesting for the capture of some number of packets via a network interface). These requests are handled by the GWT RPC mechanism, which makes calls to the server using Asynchronous JavaScript over XML (AJAX), a protocol for making server side requests that does not require the web page the user is on to refresh. When an AJAX call is made, the server handles the

request appropriately and issues a response which is handled by an asynchronous callback. The asynchronous nature of the callback allows the user interface to stay responsive to user input while the server is processing the request. When the server response is returned, the asynchronous callback is either ignored in the case of trivial actions, or it performs some function that updates the interface. For example, if a user specifies that a signature file should be generated from labeled packets, a request will be sent to the server to perform the operation. When the signature is generated, the server returns a response which is the callback then handles to update the interface showing the new signature in the list of active signatures.

The 'controller' portion of the system is responsible for handling incoming requests and manipulating and updating the model. It represents the core functional logic of the system. For our architecture, the controllers interface with the view through a series of Servlets which then calls specific service singletons to service the requests. These singletons then spawn worker threads which performs the task. The worker threads may update the underlying data model and upon completion of work the response is dispatched by the controller back to the view. Dependency injection provided by Guice is used within the controller to call and utilize the services which are responsible for dealing with user requests. The use of dependency injection helps reduce cycles in external dependencies and substantially improve the testability of the system, allowing for easy mocking of external components. Dependency injection also greatly enhances the ease with which plugins and additional functionality may be constructed by allowing run time swapping of core services. With our system, Guice managed singletons are injected into the controller at run time to provide functionality such as detection, signature generation and other tasks. These singletons are then responsible for managing worker threads which perform specific tasks attached to the singleton. For example, the 'training' service

manages threads which train the detector by processing captured packet data and converting them into detection signatures. Because the creation of threads is an expensive operation, we use thread pools to recycle and reuse threads. A thread pool is a collection of threads that are pre-allocated for use with the system. When an operation requiring a thread is performed, a thread is retrieved from the thread pool and the runnable subroutine is loaded into the thread and executed. When the routine is completed, the thread is returned to the pool where it may be reused again.

The 'model' layer of the software represents the data that is operated upon by the system. In our system, the model consists of persistent data stored in a relational database, flat files stored on the disk, as well as physical hardware such as the network interface. In order to provide an easy to maintain, 'all Java' codebase, the underlying model is manipulated through an abstraction layer which maps the physical model to Java. For hardware, this is done through the jnetpcap library which interfaces with physical hardware using JNI (Java Native Interface). Calls to the hardware are done through Java function calls which are then translated into native code. Jnetpcap allows for the rapid capturing of packets from network interfaces, including wireless interfaces. For our database, we utilized the ORM Oracle TopLink and communicate with it through the JPA2 framework. ORMs map the relational world of databases to the object oriented world of Java code. 'Entities' are Java classes which may be persisted to a database, containing data that may be stored across one or several tables. An entity manager 'manages' the connection between the object in Java land and its persistent data in the database. Under JPA2, an Entity may be 'detached', 'managed', or 'removed' from the entity manager's *persistence context*, which represents all entities currently kept track of by the entity manager. A variety of design patterns exist for using ORMs, and for our project we primarily utilize the 'session per transaction' and 'session per multiple transactions' pattern. In this pattern, a persistence entity is

‘managed’ only for the lifetime of a given transaction, or several transactions, before it is detached from the manager. This pattern was selected as the data model for our project does not require a constant connection to the database, and are only required to be updated at specific intervals.

7.3 User flow

The user workflow of the system follows four steps, moving from upload to detection.

First, a user must upload a training file. This file should be a pcap file containing malicious, or non-malicious traffic data. Alternatively, the pcap file can contain mixed data, in which case a second label file should be provided which annotates the data according to their MAC address. All traffic corresponding to the MAC address indicated in the file will be parsed as malicious, and the rest non-malicious. This functionality is handled through the File RPC Service. After uploading a file the server will generate an ID for the file from which the client can fetch through appropriate API calls. Alternatively, the user can specify a device and an interface to capture packets on for a given time in order to generate a file that way. Files generated in this way are assumed to be non-malicious.

The user can then select a set of files in order to generate a signature which will be used for detection. Signatures are generated by specifying one or more files that have been uploaded to the server. This is done through the Training RPC Service, which takes the id of the files generated and creates a signature. The signature is also associated with an id which can be used by the client to indicate to the server which signature to use for detection.

Live detection is the last step of the process. Here, the user can select a device and an interface to capture packets and perform live detection on. For now, this is limited to a single device and interface, but in the future this functionality may be expanded to include multiple devices. The

user selects a signature file to use for detection and then instructs the system to begin monitoring duties. The Detection RPC Service spawns a worker task which then monitors the traffic passing through the hardware interface and compares the live data against its signature to classify them.

7.4 Summary

In this chapter, we gave an overview of a live implementation of the system which may be run on a traditional web based platform allowing multiple users to train and perform malicious traffic detection from an interface controlled through the browser.

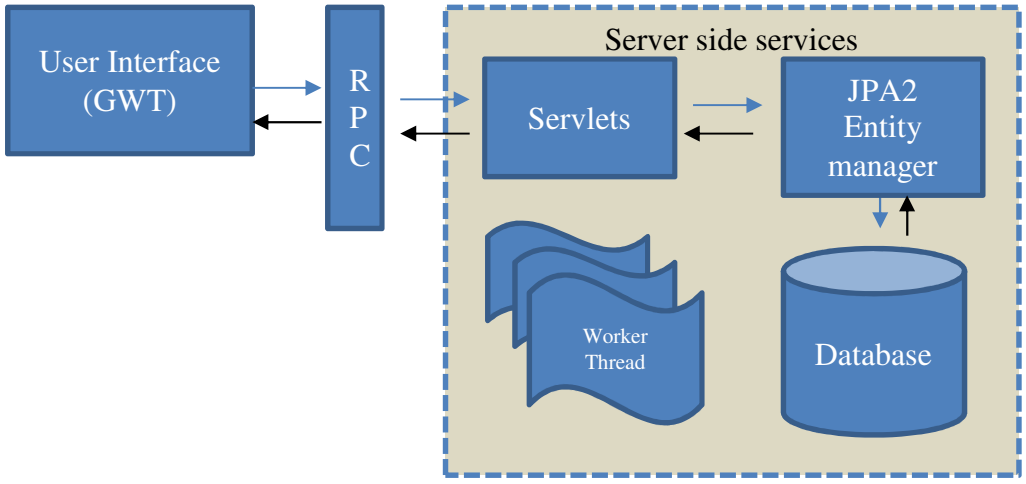

Figure 7.1: System architecture diagram

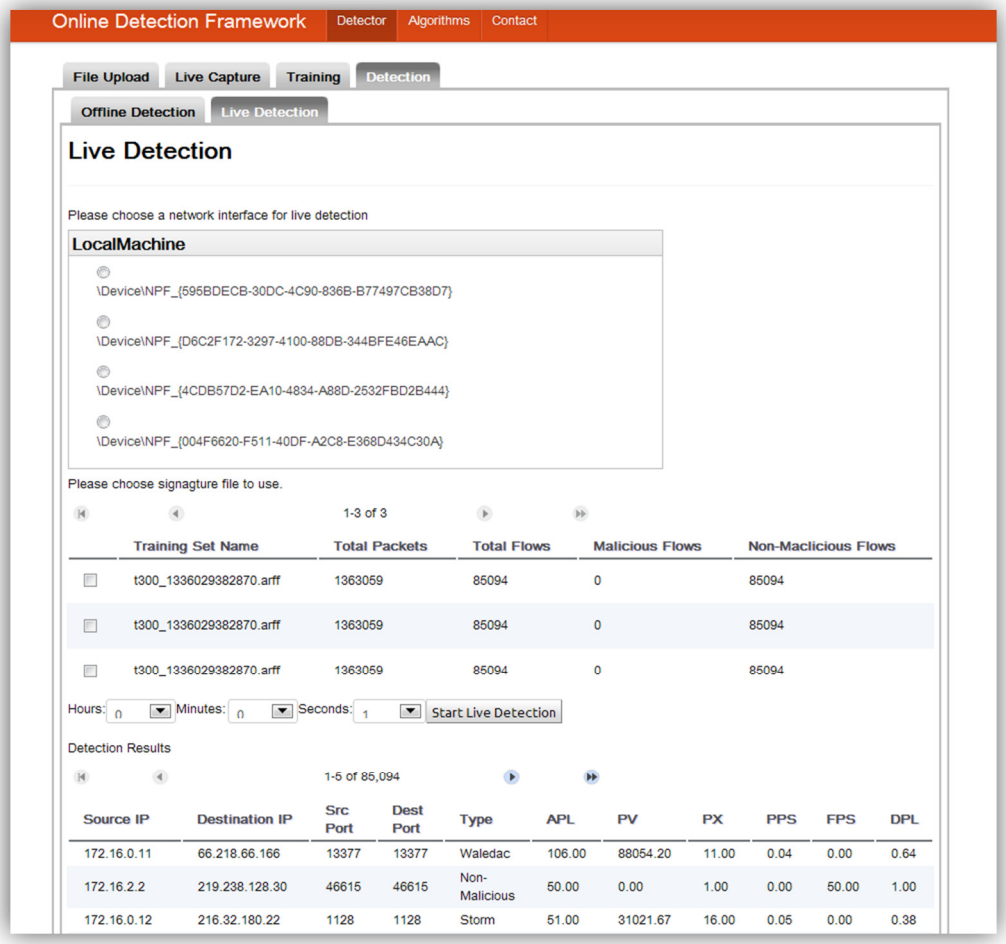

Figure. 7.2: Screenshot of system

Chapter 8

Conclusion

In this thesis we proposed a system for detecting bot activity in both the command and control and attack phases based on the observation of its network flow characteristics for specific time intervals. We emphasize the detection in the command and control phase because we would like to detect the presence of a bot before any malicious activities can be performed, and we use the concept of time intervals to limit the duration we would have to observe any particular flow before we may raise our suspicions about the nature of the traffic. We showed that using a decision tree classifier, we were able to successfully detect botnet activity with high accuracy by simply observing small portions of a full network flow, allowing us to detect and respond to botnet activity in real time. By comparing the true and false positive rates of our detector at various time window sizes, we have determined that a duration of 180 seconds provided the best accuracy of detection while a time window of 10 seconds was still able to produce an effective detector with a true positive rate of over 90% and a false positive rate under 5%.

This approach has several benefits over traditional approaches based on group behavior and anomaly detection. First, it does not require significant malicious activity to occur before detection as it can recognize command and control signals, and it does not require the group behavior of several bots before it can be confident about making a decision.

For our future work, we note that in order to create a framework that is truly robust, we must additionally produce a system which allows for the evolution of classifiers while it is still operating, without requiring an offline training process. Such a system would ideally also be capable of identifying new threats without training on existing datasets.

For our Fast Flux detector, we have demonstrated that malicious and non-malicious fast flux behavior may be classified through a machine learning process with very high accuracy. Some non-malicious sites present some problems for classification as they exhibit some behavior typically seen in malicious networks. We recommend an improvement over our existing system by adding additional confidence discriminators. One discriminator that may be of interest is the homogeneity of IP addresses within a fast flux network. As malicious networks tend to be more distributed, measuring the number of subnets members of a fast flux network belongs to could be an excellent way of eliminating false positives.

Bibliography

[1] Rajab, Moheeb Abu, et al., "A Multifaceted Approach to Understanding the Botnet Phenomenon." New York : ACM, 2006. Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. 1-59593-561-4.
[2] Grizzard, Julian B, et al., "Peer-to-Peer Botnets: Overview and Case Study." Berkeley : USENIX Association, 2007. Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets.
[3] Holz, Thorsten, et al., "Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm." Berkeley : USENIX Association, 2008. Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats.
[4] Faily, Maryam, Shahrestani, Alireza and Ramadass, Sureswaran., "A Survey of Botnet and Botnet Detection." s.l. : Third International Conference on Emerging Security Information, Systems and Technologies, 2009.
[5] The Honeynet Project., "HOW FAST-FLUX SERVICE NETWORKS WORK." *The Honeynet Project.* [Online] August 16, 2008. [Cited: October 10, 2011.] http://www.honeynet.org/node/132.
[6] Sperotto, Anna, et al., "An Overview of IP Flow-Based Intrusion Detection." *IEEE Communications Surveys & Tutorial.* 2010, Vol. 12, 3.
[7] Gu, Guofei, et al., "BotHunter: Deteting Malware Infection Through IDS-Driven Dialog Correlation." 2007. Proceedings of the 16th USENIX Security Symposium. pp. 167-182.
[8] Gu, Guofei, et al., "BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection." San Jose : Proceedings of the 17th conference on Security symposium, 2008.
[9] Yu, Xiaocong, et al., "Online Botnet Detection Based on Incremental Discrete Fourier Transform." s.l. : Journal of Networks, 2010, Issue 5, Vol. 5.
[10] Livadas, Carl, et al., "Using Machine Learning Techniques to Identify Botnet Traffic." 2006. 2nd IEEE LCN Workshop on Network Security. pp. 967-974.
[11] Wang, Binbin, et al., "Measuring Peer-to-Peer Botnets Using Control Flow Stability." Fukuoka : s.n., 2009. International Conference on Availability, Reliability and Security. p. 663. 978-1-4244-3572-2.
[12] Jiayan, Wu, et al., "A Comparative Study for Fast-Flux Service Networks Detection." Seoul : IEEE Computer Society, 2010. International Conference on Networked Computing and Advanced Information Management, NCM 2010, Sixth International Joint Conference on INC, IMS and IDC: INC 2010: International Conference on Networked Computing, IMS 2010: International Conference on Advan. pp. 346-350.
[13] Caglayan, Alper, et al., "Behavioral Patterns of Fast Flux Service Networks." 2010. Proceedings of the 43rd Hawaii International Conference on System Sciences. pp. 1 - 9. 978-1-4244-5509-6 .
[14] Al-Duwairi, Basheer and Lina, Al-Ebbini., "BotDigger: A Fuzzy Inference System for Botnet Detection." Barcelona : The Fifth International Conference on Internet and Applications and Services, 2010.
[15] Gu, G, Zhang, J and Lee, W., "BotSniffer: Detecting botnet command and control channels in network traffic." s.l. : Proceedings of the 15th Annual Network and Distributed System Security Symposium, 2008.

[16] Villamarín-Salomón, Ricardo and Brustoloni C, José., "Bayesian Bot Detection Based on DNS Traffic Similarity." Honolulu : ACM, 2009. Proceedings of the 2009 ACM symposium on Applied Computing. pp. 2035-2041. 978-1-60558-166-8 .
[17] Jun, Li, et al., "P2P Traffic Identification Technique." Harbin : s.n., 2007. International Conference on Computational Intelligence and Security. pp. 37-41. 0-7695-3072-9.
[18] Sinclair, Greg, Nunnery, Chris and Kang, Brent B., "The Waledac Protocol: The How and Why." 2009. Proceeding of 4th IEEE International Conference on Malicious and Unwanted Software.
[19] The Honeynet Project., French Chapter | The Honeynet Project. *The Honeynet Project.* [Online] http://www.honeynet.org/chapters/france.
[20] Szabó, Géza, et al., "On the validation of traffic classification algorithms." Berlin, Heidelberg : Springer-Verlag, 2008. Proceedings of the 9th international conference on Passive and active network measurement. pp. 72-81.
[21] Lawrence Berkeley National Laboratory and ICSI., LBNL/ICSI Enterprise Tracing Project. *LBNL Enterprise Trace Repository.* [Online] 2005. http://www.icir.org/enterprise-tracing.
[22] Witten, Ian H, et al., *Weka: Practical Machine Learning Tools and Techniques.* 1999.
[23] Nazario, Jose and Holz, Thorsten., "As the net Churns: Fast-Flux Botnet Observations." s.l. : IEEE Press, 2008. International Conference on Malicious and Unwanted Software (Malware). pp. 24-31.
[24] Glosser, David., "BlackHole DNS for Malware and Spyware." *DNS-BH - Malware Domain Blocklist.* [Online] November 27, 2007. [Cited: March 27, 2012.] http://www.malwaredomains.com/wordpress/?page_id=6.
[25] Leonard, Justin, Shouhuai, Xu and Sandhu, Ravi., "A Framework for Understanding Botnets." Fukuoka : Fukuoka Institute of Technology, 2009. International Workshop on Advances in Information Security.
[26] Roesch, M., "Snort - lightweight intrusion detection for networks." 1999. Proceedings of USENIX LISA'99.
[27] Yu, Sheng, Zhou, Shijie and Wang, Sha., "Fast-flux attack network identification based on agent lifespan." 2010. Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on. pp. 658-662. 978-1-4244-5850-9.
[28] Wang, Xiao, et al., "Analyzing the availability of fast-flux based service network under countermeasures." 2010. Communications, Circuits and Systems (ICCCAS), 2010 International Conference on. pp. 259-264. 978-1-4244-8224-5.
[29] Borgaonkar, Ravishankar., "An Analysis of the Asprox Botnet." 2010. Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on . pp. 148-153. 978-1-4244-7517-9 .
[30] Passerini, Emanuele, et al., "FluXOR: Detecting and Monitoring Fast-Flux Service Networks." *Detection of Intrusions and Malware, and Vulnerability Assessment.* s.l. : Springer Berlin / Heidelberg, 2008, Vol. 5137, pp. 186-206.
[31] Zhou, C.V., et al., "A Self-Healing, Self-Protecting Collaborative Intrusion Detection Architecture to Trace-Back Fast-Flux Phishing Domains." 2008. Network Operations and Management Symposium Workshops, 2008. NOMS Workshops 2008. IEEE. pp. 321-327.
[32] Caglayan, Alper, et al., "Real-Time Detection of Fast Flux Service Networks." 2009. Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications & Technology . pp. 285-292.

[33] Zhao, David, et al., "Peer to Peer Botnet Detection Based on Flow Intervals." s.l. : Springer, 2012. Proceedings of the 27th IFIP TC 11 Information Security and Privacy Conference.
[34] D. Zhao, I. Traore, “P2P Botnet Detection through Malicious Fast Flux Network Identification”, 7th International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing -3PGCIC 2012, November 12-14, 2012, Victoria, BC, Canada.