Dissertation

Detecting Advanced Botnets in Enterprise Networks

Submitted by

Han Zhang

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2017

Doctoral Committee:

Advisor: Christos Papadopoulos

Indrakshi Ray
Shrideep Pallickara
Stephen C. Hayne

Abstract

Detecting Advanced Botnets in Enterprise Networks

A botnet is a network composed of compromised computers that are controlled by a botmaster through command and control (C&C) channel. Botnets are more destructive compared to common virus and malware, because they control the resources from many compromised computers. Botnets provide a very important platform for attacks, such as Distributed Denial-of-Service (DDoS), spamming, scanning, and many more. To foil detection systems, botnets began to use various evasion techniques, including encrypted communications, dynamically generated C&C domains, and more. We call such botnets that use evasion techniques as advanced botnets. In this dissertation, we introduce various algorithms and systems to detect advanced botnets in enterprise-like network environment.

Encrypted botnets introduce several problems to detection. First, to enable research in detecting encrypted botnets, researchers need samples of encrypted botnet traces with ground truth, which are very hard to get. Traces that are available are not customizable, which prevents testing under various controlled scenarios. To address this problem we introduce BotTalker, a tool that can be used to generate customized encrypted botnet communication traffic. BotTalker emulates the actions a bot would take to encrypt communication. To the best of our knowledge, BotTalker is the first work that provides users customized encrypted botnet traffic.

The second problem introduced by encrypted botnets is that Deep Packet Inspection (DPI)-based security systems are foiled. We measure the effects of encryption on three security systems, including Snort, Suricata and BotHunter (BH) using the encrypted botnet traffic generated by BotTalker. The results show that encryption foils these systems greatly. Then, we introduce a method to detect encrypted botnet traffic based on the fact that encryption increases data's entropy. In particular, we present two high-entropy (HE) classifiers and add one of them to enhance BH by utilizing the other detectors it provides. By doing this HE classifier restores BH's ability to detect bots, even when they use encryption.

Entropy calculation at line speed is expensive, especially when the flows are very long. To deal with this issue, we introduce two algorithms to classify flows as HE by looking at only part of a flow. In particular, we classify a flow as HE or low entropy (LE) by only considering the first M packets of the flow. These early HE classifiers are used in two ways: (a) to improve the speed of bot detection tools, and (b) as a filter to reduce the load on an Intrusion Detection System (IDS). We implement the filter as a preprocessor in Snort. The results show that by using the first 15 packets of a flow the traffic delivered to IDS is reduced by more than 50% while maintaining more than 99.9% of the original alerts. Comparing our traffic reduction scheme with other work we find that they need to inspect at least 13 times more packets than ours or they miss about 70 times of the alerts.

To improve the resiliency of communication between bots and C&C servers, bot masters began utilizing Domain Generation Algorithms (DGA). DGA technique avoids static blacklists as well as prevents security specialists from registering the C&C domain before the botmaster. We introduce BotDigger, a system that detects DGA-based bots using DNS traffic without a priori knowledge of the domain generation algorithm. BotDigger utilizes

a chain of evidence, including quantity, temporal and linguistic evidence to detect an individual bot by only monitoring traffic at the DNS servers of a single network. We evaluate BotDigger's performance using traces from two DGA-based botnets: Kraken and Conflicker, as well as a one-week DNS trace captured from our university and three traces collected from our research lab. Our results show that BotDigger detects all the Kraken bots and 99.8% of Conficker bots with very low false positives.

Acknowledgements

During my Ph.D. study, I have received support, encouragement, and help from many people. Without them, it would not have been possible to finish this work.

My sincere and deep gratitude goes to my advisor, Dr. Christos Papadopoulos for his patient guidance and consistent support. On the academic journey, is is easy to get lost and go to wrong direction. He brought me into science and research, led me on this journey, and provided me with insightful advice during my entire Ph.D. study.

I would also like to thank other members of my committee, Dr. Indrakshi Ray, Dr. Shrideep Pallickara, and Dr. Stephen C. Hayne. They provided me with insightful comments on my research exam, preliminary exam and final exam, as well as helped me improve the quality of my work.

I have been also very lucky to be a member of a great research group - The Network Security Research Group. Discussions with my colleagues always inspired me new ideas on my work.

I would particularly like to thank my family, especially my mother, Yan Zhang. She inspired my interests in mathematics when I was a child. She has taught me a lot and she always encourages me on my way of exploring new things. Finally, I would like to thank my wife, Zhuo Zhao, for her kindness and optimism. Her smile always gives me great courage to overcome any obstacles. Her love makes this happen.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1. Botnets Overview

Internet has entirely changed the world in the last few decades. However, the tremendous growth of Internet brought many cyber attacks. Cyber security constitutes one of the most serious threats to the current society, costing hundreds of billions of dollars each year [16]. In 2015, 318 data breaches happened, and 9 of them exposed more than 10 million identities, 429 million identities were exposed in total [17]. The same year, approximately 100 billion spam were sent every day; 431 million new malware variants were added; 362,000 Crypto-Ransomware were reported; and 1 in 3,172 websites were found with malware. In October 2016, a large DDoS attack was aimed at Dyn [5], an Internet infrastructure company, impacting access to a lot of companies in U.S., including PayPal, Twitter, GitHub, Amazon, Netflix, and many more. One important question we would ask is that where these attacks come from. The answer is that botnets provide platforms for them [87].

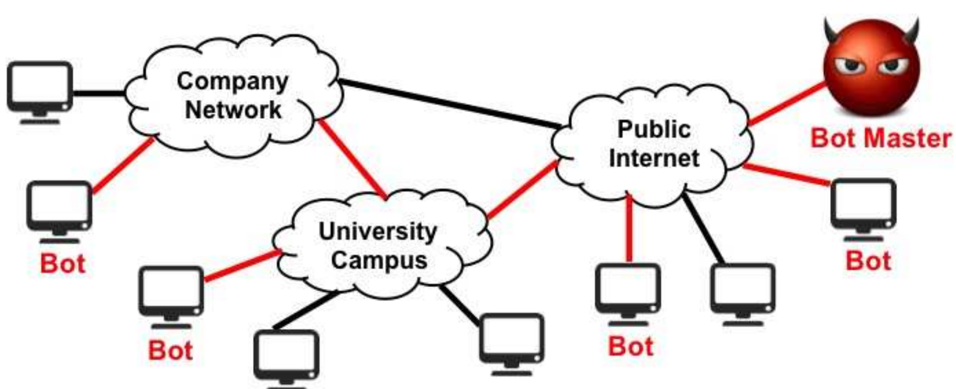

Figure 1.1. Botnet Example

Botnet is a network composed of infected computers named bots, and all these bots are controlled by a botmaster. An example of botnet is shown in Figure 1.1. Bots do not have to be in the same network, instead, they can distribute in multiple networks. All the bots follow the same commands from the botmaster via commands and control (C&C) channel, shown in red lines in the figure. Botnets are more destructive compared to common virus and malware, because they control the resources from a great number of compromised computers. In addition, bots are coordinated by the botmaster, so the attacks can be well designed and performed simultaneously. Botnets can be used to perform many attacks, we list some examples of botnets and the corresponding attacks in Table 1.1.

TABLE 1.1. Botnets and Attacks

Attack	Botnets
DDoS	BlackEnergy, MrBlack, Nitol, Metulji, Mariposa
Spam	Conficker, Kraken, Rustock, Waldec, Grum, Windigo, Cutwail, Srizbi, Storm, Bobax, Ozdok, OneWordSub, Nucrypt, Wopla
Phishing and Pharming	Asprox, Tequila
Click Fraud	ZeroAccess, Chameleon
Ransomware	Vundo, Gameover ZeuS
Steal Data	Sinowal, Alureon, Gameover ZeuS
Install Malware	Monkif, Rimecud, Pushbot

Attackers can use botnets to make money in many ways. For example, they can steal victims' personal and business information and sell them, perform DDoS attacks to blackmail enterprises, encrypt victims' files and ask for ransom. Another common way to make money is advertised-spamming. Spam-advertised sites are built on the affiliate programs, which provide retail contents and back-end services to a set of client affiliates. The affiliates are responsible to attract the customers in any advertising way and they get paid on a commission basis. In [48], Kanich et al. estimate that seven counterfeit pharmacies and three counterfeit software stores have more than 82,000 and 37,000 orders, producing revenues of 5.9 million and 3.9 million, respectively. Besides, botnets can also make profits through Pay-Per-Install (PPI) service. In PPI service, a botmaster can be the PPI provider who installs client provided malware on compromised computers and charges the clients based on the number of successful installations. An attacker can easily build his own botnet by buying installations from PPI providers with very low cost. The installation fees vary from $100 to $180 for a thousand unique successful installation in the most popular regions but only $7 to $8 in the least demanded regions [27].

We want to emphasize that besides malicious bots, there are some legitimate bots. Assuming a group of enterprises buy the same service from a network security company. With approved permission, the security company may collect information from the monitors in each enterprise, and then make decisions based on the combined information. In addition, the security company also sends updates to the machines in enterprises. In this scenario, the machines installing the security software and the security company can be considered as legitimate "bots" and "botmaster", respectively. In this dissertation, we do not consider such legitimate botnets, but only focus on the malicious botnets composed of ***compromised*** computers.

1.2. Botnet Architecture

One important component in botnet is the C&C channel, which is used to exchange commands between bots and botmaster. Based on the C&C protocol, the botnets can be classified into centralized and decentralized.

1.2.1. Centralized Botnets. In a centralized botnet, there is a central node between the botmaster and bots, the botmaster publishes commands to this central node and then this central node sends the commands to the bots or the bots fetch the commands from it. IRC-based (Internet Relay Chat) and HTTP-based botnets are the two main types of centralized botnets. In IRC-based botnets, the botmaster and bots connect to the same IRC server. Once the botmaster sends a command in the channel, the bots connected to the same channel can receive it. IRC-based botnets include Rbot, Spybot, Agobot, SDBot, GT Bot [38], [23]. In HTTP-based botnets, botmaster publishes commands on a HTTP server, and then the bots periodically query the server to fetch the commands. Compared to IRC-based botnets, it is harder to detect and block HTTP-based botnet traffic. First, the HTTP-based C&C traffic is hidden in a great amount of legitimate HTTP traffic, and analyzing all these traffic is time consuming. Second, we cannot block HTTP traffic because it is used by many legitimate applications. HTTP-based botnets include Kraken [75], Conficker [53], [52], Bamital [3], Torpig [87], Srizbi [95], Murofet [78].

1.2.2. Decentralized Botnets. Different from centralized botnets, there is no central server in a decentralized botnet and it uses peer-to-peer technology to exchange commands, so it is also named P2P botnet. P2P botnets include Storm [71], Nugache [88], Waledac [80] and Zbot [60]. In recent years, decentralized botnet is getting more popular because they are hard to detect and shut down. In centralized botnets, there is a central server to publish the commands, when this central server is detected and cleaned, the entire botnet is shut down because the bots cannot receive the commands any more. On the contrary, decentralized botnets do not rely on a single server to exchange information.

1.3. Advanced Botnets

The arms race between botnets and security specialists has lasted for years. On one hand, security specialists spend lots of time and money on inventing botnet detection systems. On the other hand, botnets utilize various techniques to become more robust and stealthy, evading existing detection systems. In the context of this dissertation, we call the botnets that use evasion techniques as ***Advanced Botnets***.

Botnets pay much attention on evolving and protecting the C&C channels, because once the C&C channels are detected and blocked, the botmaster will lose the control of its entire army.

To evade DPI-based detection systems, bots began using encrypted C&C channels: Storm utilizes XOR for encrypted communications [45], [71]; Waledac's communication is encrypted by AES [80]; all the C&C traffic in Rustock is encrypted using RC4 [29]. In [93], Wang et al. design an advanced hybrid peer-to-peer botnet with several new features to make it hard to detect. One of these features is that each bot randomly generates a symmetric encryption key for communication. In [86], Stinson et al. evaluate the evadability of botnet detection systems, and present several approaches a botnet can use to defeat detection systems. One of the approaches encrypts connections between the botmaster and bots and among the bots themselves.

Many protocols can be utilized as C&C channels, such as IRC, HTTP, OVERNET [45], and more. Although many botnets began to use P2P protocols as C&C channels, HTTP-based botnets are still very popular because they are easy to implement and maintain. In the early years, the C&C domain was hard-coded in binary, introducing a single point of failure. Once the security specialists obtain the C&C domain (e.g., running the bot binary in sandbox and analyzing the traffic), the domain can be blacklisted. To become more robust, botnets generate C&C domains dynamically on the fly using *Domain Generation Algorithm* (DGA), which foils the static blacklists. They are named DGA-based botnets. In particular, hundreds or thousands of domains can be algorithmically generated every day, but the botmaster only registers one or a few of them as C&C domains and publishes the commands there. DGA technique has been used in many botnets, such as Kraken [75], Conficker [53], [52], Bamital [3].

1.4. Problem Statement

Encrypted botnet traffic makes detection much harder because many security systems rely on DPI techniques. Encryption also makes it very hard to get ground truth necessary to develop detection algorithms. While bot communication can be captured in sandboxes from binaries harvested around the net, traces collected from sandboxes are fairly static. Researchers would like to adjust parameters such as encryption type, timing, etc., the same way a bot might act to alter its signature and foil detection. Another disadvantage of using captured encrypted botnet traffic is that the researchers do not know which part of traffic is encrypted, nor the traffic is encrypted by which encryption algorithm. Consequently, it is important to provide a tool that researchers can use to generate customized encrypted botnet traffic based on unencrypted botnet traffic.

It is true that encryption foils DPI techniques, but it is different from that encryption makes the DPI-based security systems totally blind. The reason is that some systems are composed of multiple detection engines, and only one or a few of them reply on DPI techniques, while the others may utilize hosts' behaviors for detection. For example, as we show in Section 5, BotHunter is a botnet detection system composed of eight malicious event detectors. Although the two detectors that rely on DPI are foiled by encryption, the other six that do not rely on DPI are still able to detect some (not all) encrypted botnet traffic. Consequently, we need to measure the damage result from encrypted botnet traffic on security systems. After evaluating three security systems that rely on DPI, we find that they are significantly foiled by encrypted botnet traffic. It is therefore important to develop new systems to detect encrypted botnet traffic, especially in the early stage.

Although DPI-based IDS is foiled by encrypted communications, it still plays an important role in protecting enterprise networks since many botnets and malware do not use encryption. However, the overhead of DPI can be substantial: in [66], Namjoshi et al. indicate that in the worst case it may take up to one second to find a match for the regular expressions used in IDS. This computationally intensive analysis for every packet introduces performance issues when the IDS is deployed on high speed links. Delivering certain content, such as video, audio, encrypted or compressed data to IDS is a waste of computation resources, since the IDS can not understand such content. As a result, we need to classify the traffic into two groups. One group contains the traffic that can be understood by IDS, while the other includes the traffic that cannot be understood. Then, only the traffic in the former group is delivered to IDS. Note that we need to classify the traffic as early as possible. If we make the decision at the end of a flow, all the packets will have already been delivered to IDS, and the load on IDS is not reduced at all. Reducing the traffic to IDS does not directly detect encrypted botnet traffic, but it helps save valuable computation resources. The saved resources can be used to detect bots that use evasion techniques other than encryption, for example, DGA-based bots. Consequently, reducing the load on IDS indirectly helps detect advanced botnets.

Besides encryption, botnets also use DGA technique for evasion. The main aim of using DGA technique is to avoid static blacklists and single point of failure. In addition, another advantage of using DGA technique is to prevent security specialists from registering the C&C domain before the botmaster due to two reasons. First, learning the domain generation algorithm requires reverse engineering of the bot binary, which is time consuming. In addition, the botmaster can patch the bots and change the algorithm at any time, then the binary has to be analyzed again. Second, even if the algorithm is successfully extracted, as the bots usually randomly query the generated domains and botmaster can register any of them, so the security specialists have to register many of the generated domains to catch the botnet communications, costing lots of money. As a result, it is important to detect DGA-based botnets. There are four reasons to detect DGA-based botnets using DNS traffic. First, the DGA bots cannot get rid of DNS queries as they have to get the IP addresses of C&C domains. Second, the amount of DNS traffic is comparable much less than the overall traffic. Third, the DNS traffic of DGA bots has different patterns from legitimate one. Fourth, detecting bots based on DNS traffic enables us to stop the bots even before they perform attacks. Consequently, we decide to introduce a method to detect DGA-based bots using DNS traffic.

1.5. Research Contributions

In this dissertation, we introduce several algorithms and systems to detect advanced botnets in enterprise networks. An enterprise networks can be a company network, a university network, or a local area network (LAN). This dissertation makes the following specific contributions:

(1) **BotTalker: generating encrypted, customizable C&C Traces**

We design and implement BotTalker, the first work that provides user customized encrypted botnet traffic converted from real botnet traffic, rather than simulated traffic. BotTalker contributes to the community in the following ways.

- Benefits IDS Developers: BotTalker provides IDS developers the ability to generate customized encrypted botnet traffic from real botnet traces to test their detection algorithms.
- Benefits IDS Customers: BotTalker helps customers (e.g., network administrators) evaluate a new IDS and make informed decisions. Customers test a new IDS by blending background traffic collected from their network with encrypted bot traffic generated by BotTalker, and then feed the traffic to the new IDS for performance testing.
- Benchmark for IDS: Currently the detection systems use their own data sets for evaluation and the data sets are not public. It is impossible to compare the detection systems' performance directly. Since BotTalker is public it can be used as a benchmark to evaluate different IDSs.

(2) **Measuring the damage result from encryption on security systems**

- Using BotTalker, we evaluate the effect of encrypted botnet traffic on three common security systems - BotHunter, Snort and Suricata. The results show that encrypted traffic foils these systems greatly.

(3) **Detecting encrypted botnet traffic using high entropy classifiers**

- We show that encryption produces high entropy (HE) flows.

- We introduce two HE classifiers to distinguish HE flows from low entropy (LE) flows. The classifiers are robust to LE packets at the beginning of a flow, feature common to many security protocols.
- We extend BotHunter with the HE classifier and restore its ability to detect bots when they use encryption.

(4) **Early high entropy classifiers and traffic reduction**

- We introduce two early HE classifiers that can label HE flows by looking at only part of a flow. The results show that only 2.38% to 7.78% of the packets require entropy calculation.
- We enhance BotHunter with the early HE classifier to detect encrypted botnet traffic.
- We develop a traffic filter using the early HE classifier and implement it as a Snort preprocessor. The results show that the filter reduces the amount of traffic delivered to Snort by more than 50%, while preserving more than 99.9% of the alerts. We compare our traffic reduction method with similar work and find that they must inspect at least 13 times more packets or they miss at least 70 times of the alerts.

(5) **BotDigger: detecting DGA bots in a single network**

- We introduce a chain of evidences, including quantity evidence, temporal evidence and linguistic evidence, to detect DGA-based botnets.
- We introduce and implement BotDigger, a system that detects an individual DGA-based bot using DNS traffic collected in a single network.

- We evaluate BotDigger with two datasets from our university and lab, as well as two DGA-based botnets. The results show that BotDigger detects more than 99.8% of the bots with very low false positives.

1.6. Dissertation Organization

This dissertation is organized as follows. Background and related work are discussed in Chapter 2. Chapter 3 describes the datasets of network traffic and static application content used in our experiments. In Chapter 4, we present BotTalker, a tool that can generate encrypted and customizable C&C traces. Next, the damage result from encrypted botnet traffic on two widely used intrusion detection systems and one botnet detection system is evaluated in Chapter 5. In Chapter 6, we build two classifiers to identify high entropy flows, and use them to extend BotHunter, making it be able to detect encrypted botnet traffic. In Chapter 7, we introduce two algorithms to detect high entropy flows in the early stage, and use them to reduce the load on IDS. After that, a system named BotDigger to detect DGA-based botnets using DNS traffic is presented in Chapter 8. Finally, we conclude the dissertation and discuss future work in Chapter 9.

CHAPTER 2

Background and Related Work

In this chapter, we first discuss the existing botnet detection systems and propose a taxonomy of these systems, categorizing them into two groups. The first detects botnets by focusing on C&C traffic, and the second utilizes distinguished botnet behaviors for detection. After that, we discuss the previous work related to detecting encrypted botnet traffic, detecting DGA-based botnet and reducing load on IDS.

2.1. C&C Traffic-based Detection Systems

Many systems make use of C&C traffic for detection because there are several advantages by doing so. First, the botmaster will lose its entire army if we can detect and block the C&C traffic. Second, if the C&C traffic can be observed, we can generate signatures and use them to extract more hidden bots or use them to detect bots later. These detection systems can be categorized into three groups in further based on the C&C type, including i) the systems that can only detect centralized botnets, ii) the systems that can only detect decentralized botnets, iii) the systems that are independent of botnet architecture.

2.1.1. The systems that can only detect centralized botnets.

In [42], Gu et al. introduce BotProbe, a system detects IRC-based botnets using active probe techniques. BotProbe is a *Turing Test* that tests whether an IRC client is a bot or human based on the fact that bots have strong correlations between given C&C messages and the responses. In particular, bots are preprogrammed to respond to a set of predefined commands, and these responses are consistent for command repetition. One advantage of BotProbe is that it can detect obscure C&C messages, for example, some botnets use foreign language or XOR encryption, which fail the signature-based detection.

In [43], Gu et al. propose BotSniffer, a system that detects both IRC-based and HTTP-based botnets. BotSniffer is built on the observation that a group of bots have stronger synchronization in responses than normal network services. In particular, after the bots receive commands, they need to respond messages to the botmaster or perform corresponding attacks. If there are multiple bots that belong to the same botnet responding to the same commands, most of them are likely to respond in a similar way (e.g., bots send similar messages or perform similar activities in a similar time window). On the contrary, for most normal services, it is unlikely that they perform similar responses at similar times.

2.1.2. The systems that can only detect decentralized botnets. Decentralized (P2P) botnets are becoming more popular in recent years. There are two challenges in detecting P2P botnets, one is extracting P2P traffic from the background traffic, and the other is distinguishing botnet P2P traffic from legitimate P2P traffic after we extract P2P traffic. In [65] and [101], the authors introduce techniques to detect P2P botnets by targeting at each of these two challenges, respectively.

Extracting P2P traffic from non-P2P traffic is hard because most P2P applications use random port numbers, and some of them use encryption. In [65], Nagaraja et al. introduce an algorithm named BotGrep to extract P2P traffic based on the fact that P2P network topologies are much more structured than the others. After extracting P2P topologies, the authors use misuse detection approaches, such as honeypots and blacklists, to distinguish botnet traffic from legitimate traffic. However, if some hosts listed on blacklist belong to both P2P botnet and legitimate P2P networks, then the legitimate P2P network is also considered as botnet related.

In [101], Yen et al. present a method to distinguish botnet P2P traffic from legitimate P2P traffic. Although P2P botnet and legitimate applications are implemented on top of similar protocols, they are used for totally different purposes, so there are some differences in their communication patterns. In particular, three characterizing differences are utilized, including amount of traffic, peer churn, and timing information. One limitation of this algorithm is that it has to extract P2P traffic before applying the above three features. It would be a good idea to combine the above two algorithms, using *BotGrep* to extract P2P traffic and then using the latter technique to distinguish botnet P2P traffic from legitimate P2P traffic.

2.1.3. The systems that are independent of botnet architecture. A system to detect botnet without any prior knowledge about C&C communication is introduced by Wurzinger et al [96]. In particular, the authors run bot binaries in controlled environment and capture the traffic. Then, they look for the activities that indicate a response corresponded to occurred C&C communications. After that, they scan the traffic around the responses to detect C&C communications. However, this method requires bot binaries, so it is only able to detect existing botnets.

In [40], Gu et al. introduce BotMiner, a botnet detection system independent of botnet architecture and it does not require bot binaries and prior C&C knowledge. BotMiner is designed based on the fact that the bots belonging to the same botnet perform similar malicious activities as well as have similar communication patterns. However, BotMiner requires multiple bots for detection because it relies on the similarity in bots' communications and activity patterns.

2.1.4. Discussion of C&C Traffic-based Detection Systems. Three categories including six different C&C traffic-based detection algorithms have been described above. A crucial question to ask is that which is the best one. To answer this question, we introduce the following seven criteria to compare these systems.

(1) Botnet architecture:

This criterion indicates which type of botnet can be detected. For this criterion, the systems that are independent of botnet structure are preferable.

(2) Passive or active monitoring:

The passive monitoring algorithm does not add additional traffic to network and only monitors the traffic as it passes by. On the contrary, active monitoring algorithm injects packets into network or sends packets to servers and applications, then watches how they respond. For this criterion, passive monitoring is better than active monitoring because the latter introduces additional traffic and may affect other intrusion detection systems or traffic measurements.

(3) DPI technique is used or not:

DPI technique is commonly used to search for intrusion, viruses or user defined rules by examining the contents. However, it fails to work when the traffic is encrypted. For this criterion, the systems that do not rely on DPI technique are better.

(4) Detect a single bot or not:

This criterion indicates whether the system can detect single bot or it needs a group of bots for detection. Many detection systems that use correlations between bots need multiple bots belonging to the same family for detection. For this criterion, the systems that that can detect single bot are better.

(5) Blacklist is used or not:

This criterion indicates whether blacklist is used for detection. For this criterion, the systems that do not rely on blacklists are better because botnets can easily evade blacklists.

(6) Bot binary is used or not:

This criterion indicates whether bot binary is used for detection, for example, bot binary is run in controlled environment and the traffic is captured for analysis. One limitation of the systems that need bot binaries is that they can only detect existing botnets. For this criterion, the systems that do not rely on binaries are preferable.

(7) When to detect bots:

This criterion indicates when the system can detect the bots. For this criterion, the systems that can detect bots before attacks are preferable, followed by the ones during attacks and after attacks.

We compare the six botnet detection systems in Table 2.1 using the above seven criteria. From the table we can find that there is no detection system is better than all the others in all the seven aspects. This is not a surprising conclusion as there is no silver bullet for botnet detection, and the botnets are evolving themselves for evasion all the time.

Table 2.1. Comparison of C&C Traffic-based Detection Systems

	Bot Probe [42]	Bot Sniffer [43]	Bot Grep [65]	Plotter Trader [101]	Auto Model [96]	Bot Miner [40]
Botnet Architecture	IRC	Centralized	Decentralized	Decentralized	Independent	Independent
Passive/ Active	Active	Passive	Passive	Passive	Passive	Passive
DPI	Yes	Yes	No	No	Yes	Yes
Number of Bots	Single	Group	Group	Group	Single	Group
Blacklist	No	No	Yes	No	No	No
Binary	No	No	No	No	Yes	No
Detection Time	Before & During Attack	After Attack	After Attack	After Attack	During Attack	After Attack

2.2. Bot Activity-based Detection Systems

In the previous section, we have described the systems that focus on C&C traffic for detection. Besides the C&C traffic, many distinguishing behaviors of attacks can also be used to detect botnets. We name them bot activity-based systems. The bot activity-based systems can be categorized into three groups, including the systems that detect bots before attack, during attack and after attack.

2.2.1. Detecting Bots Before Attack. In [72], Ramachandran et al. introduce a system to detect spamming bots before the attacks based on DNS blacklist (DNSBL). DNSBL is used by many Internet services to track IP addresses that have sent spam, so the future spam from these addresses can be rejected. Besides the legitimate services, botnets themselves also send query DNSBL to check whether they are on the list. These queries from botnets are named *reconnaissance queries*. This system detects bots by taking use of two features of these reconnaissance queries, named *Spatial Relationship* and *Temporal*

Relationship. Spatial relationship indicates that the legitimate mail servers not only perform queries for others but are also queried by other mail servers. On the contrary, the bots that perform reconnaissance queries only perform queries but are not queried by others. Temporal relationship assumes the DNSBL queries from legitimate mail servers reflect actual arrival patterns of real emails. For example, the legitimate DNSBL queries from real emails tend to be diurnal.

2.2.2. Detecting Bots During Attack. In [41], Gu et al. introduce BotHunter, a system detects the botnets during the attack. BotHunter is the first real-time system that can automatically derive a profile of the entire bot detection process. The main idea is that bots should obey a sequence of suspicious events in loose order, named *bot infection dialog model* composed of eight events. BotHunter scores hosts engaging in these activities and flags them as bots if the score exceeds a threshold.

2.2.3. Detecting Bots After Attack. In [90], Stringhini et al. propose BotMagnifier, a system that learns spamming bots' behavior patterns from a set of captured bots, and then uses these patterns to extract additional bots by analyzing email transaction logs. The assumption of BotMagnifier is that bots belonging to the same botnet share similar email contents and the domains that they are sending spam to. Although this algorithm can only detect spamming bots after attacks, it can extract a great amount of hidden bots every day.

2.2.4. Discussion of Bot Activity-based Algorithms. We compare the above three bot activity-based detection systems in Table 2.2 using the seven criteria defined in Section 2.1.4. From the table we can see that there is no detection system is better than all the others in all the seven aspects.

Table 2.2. Comparison of Bot Activity-based Detection Systems

	DNSBL [72]	BotHunter [41]	BotMagnifier [90]
Botnet Architecture	Independent	Independent	Independent
Passive/Active	Passive	Passive	Passive
DPI	No	Yes	Yes
Number of Bots	Single	Single	Group
Blacklist	Yes	No	Yes
Binary	No	No	No
Detection Time	Before Attack	During Attack	After Attack

2.3. Network Traffic Generator

When we introduce a new security system, it is necessary to evaluate its false positives and true positives. Most of the introduced security systems use their own private datasets for evaluation and the datasets are not public. A widely used public dataset is the *Lincoln Laboratory Darpa Dataset* [59], [58], [44]. However, this dataset was introduced in 1998 while attack behaviors have been changed greatly, and many new attacks have been introduced in the last two decades. Besides, the Darpa dataset is static, but the researchers may want to customize the dataset. To address these needs, many techniques have been introduced to generate network traffic.

In [83], Sommers et al. introduce Malicious trAffic Composition Environment (MACE) - a framework that aims at providing basic building blocks that can be used to compose both existing and user customized attacks. MACE is composed of exploit model, obfuscation model, propagation model and background traffic model. These models enable users to generate a large set of attacks and also benign background traffic. In [79], Shiravi et al. propose a systematic approach to generate benchmark datasets for intrusion detection. The approach uses two profiles to generate traffic. The first profile describes the attacks using a attack description language, for example, ADeLe [64]. The second profile characterizes the distribution and behaviors of normal traffic. In [22], Barford et al. introduce Scalable URL Reference Generator (SURGE) to simulate a group of uses that access a Web server by applying a set of observations of Web server usage.

Besides the IDSs that use packet level information for detection, many IDSs detect malicious activities relying on flow level traffic [50], [84], [89]. These systems need flow-level data for testing, evaluating and benchmarking. To address the needs for these systems, Sperotto et al. provide a labeled flow-based data set [85]. In [81], Sommers et al. present Harpoon - an application-independent tool for generating flow level traffic. The most distinguished feature of Harpoon is that it can self-configure by automatically extracting parameters from standard Netflow logs or packet traces. Then, in [82], Sommers et al. propose another flow record generation tool named *fs*. fs aims at generating flow export records and also basic SNMP-like router interface counts (e.g., byte, packet) in an efficient way, thus the flow records of a large network topologies can be generated.

Some researchers focus on generating botnet traffic. In [21], Barford et al. introduce Botnet Evaluation Environment (BEE). BBE is a OS/Bot image that can be run on machines and it aims at providing ground truth behaviors of the bots by repeatedly running the bot binaries and changing configurations. Our work - BotTalker, is most closely related to SLINGbot [46], which is a tool that can generate user customized C&C traffic. There are several differences, however, between SLINGbot and BotTalker. First, SLINGbot does not emulate encrypted botnet communications although it supports certificates and password. Second, SLINGbot only generates C&C traffic. Besides the C&C traffic, the traces provided by BotTalker also include other traffic (e.g., propagation traffic), which can be used by detection systems, like BotHunter. In [54], Lee et al. propose Rubot - a framework that allows researchers to build bot code and then deploy the botnet to a closed network. However, the bots' behaviors and communications are emulated, which could be different from the real ones. The most significant difference between BotTalker and all the above efforts is that they generate emulated traffic while BotTalker converts real C&C traffic to encrypted traffic. BotTalker preserves all the network level characteristics (timing information, packet size, etc.), which is important because many detection systems make decisions based on them.

2.4. Encrypted Data Detection

An important feature of encrypted data is that it has high entropy. Consequently, entropy has been widely used to detect encrypted data in previous work. Lyda et al. use entropy to detect encrypted and packed malware based on the findings that their entropy is higher than the entropy of native executables [61]. Malhotra et al. focus on detecting egress encrypted communications by using various statistical methods, including Kolmogorov-Smirnov test, arithmetic mean, index of coincidence test, information entropy and more [63]. However, the approach does not work in the case where the flows begin with unencrypted packets, followed by encrypted contents. Besides, the approach is only evaluated using offline files, but not real network traffic. In [33], Dietrich et al. find that Feederbot uses DNS to carry their C&C traffic, and the traffic is encrypted using RC4. Then, they introduce an approach to detect such C&C traffic using entropy.

In [39], Olivain et al. introduce a method to classify data as high entropy (HE) and use it to detect subverted traffic in encrypted communications. In particular, thresholds for various lengths of data are calculated. The entropy of a chunk data is compared with the corresponding threshold to decide the data is HE or low entropy (LE). Based on the definition of HE and LE data introduced by Olivain, Dorfinger et al. attempt to reduce the amount of traffic passed to a Skype detector in real time in [37]. The assumption they make is that Skype traffic is encrypted and the first packet of the flow should be HE. As a result, the flow is considered as non-Skype if the first packet is LE. Then, the authors use a similar approach to classify traffic as encrypted and unencrypted in real time by considering whether the first packet of a flow is HE or LE [36], [35]. However, these approaches only work for the protocols in which the encrypted conversion begins from the first packet. Although many protocols encrypt data, they exchange packets at the beginning of a flow that are not encrypted. After the initial exchange, subsequent packets are encrypted. In this case, the proposed approaches will not work. On the contrary, our HE classifiers and early HE classifiers are able to skip the unencrypted part and detect the following encrypted communications.

In [74], Rossow et al. introduce PROVEX to detect encrypted botnet traffic. In particular, PROVEX first collects botnet binaries and obtain a set of keys or key extraction functions from them with the help of sandboxes and reverse engineering. After that, PROVEX decrypts the packets on the network to detect encrypted C&C communications. PROVEX can only detect the bots whose keys are extracted. Besides, extracting keys and decrypting all the traffic is computationally expensive. On the contrary, out method detects potential encrypted traffic without decryption and it is independent of encryption algorithms.

2.5. DGA-based Botnet Detection

Among the works that detect malicious domains (e.g., C&C domains, phishing pages, etc.) and DGA-based botnets, many of them share similar assumptions including 1) domains generated by the same DGA have similar attributes, 2) DGA bots have different DNS traffic patterns than legitimate hosts.

In [25], Bilge et al. introduce EXPOSURE to find malicious domains. In particular, they first extract 15 features from a domain and all the IPs that the domain is mapped to. Then, they build a decision tree from training set. After that, any given domain can be classified as malicious or legitimate using the tree.

In [18], Antonakakis et al. introduce a dynamic reputation system for DNS named Notos. Notos builds models of known legitimate and malicious domains using 18 network-based features, 17 zone-based features, and 6 evidence-based features. The legitimate information includes DNS information collected from multiple recursive DNS resolvers. Malicious information includes malicious domains and IPs obtained from multiple sources. Then, these models can be used to compute a reputation score for a new domain.

Antonakakis et al. introduce Pleiades in [20]. Pleiades uses NXDomains collected from local recursive DNS (RDNS) servers to detect DGA-based bots by using both clustering and classification algorithms. The clustering algorithm is used to find large clusters of NXDomains that (i) have similar linguistic attributes, and (ii) are queried by multiple bots during a given time interval. The classification algorithm is used to assign the generated clusters to known DGA models.

In [19], Antonakakis et al. introduce Kopis, a system monitors upper level DNS traffic collected from authoritative name servers or top level domain (TLD) servers to detect malware domains based on global view of DNS query resolution patterns. Kopis first analyzes upper level DNS traffic and extracts three groups of statistical features, including requester diversity, requester profile and resolved-IPs reputation. Then, Kopis builds a statistical classification model from a given training set of known legitimate and malicious domains. After that, this statistical classification model can be used to label a new domain as legitimate or malicious.

In [91], Villamarn-Salomn et al. propose a Bayesian approach to detect bots based on the assumption that bots in the same botnet have similar DNS traffic patterns. In [47], Jiang et al. introduce a method to detect suspicious activities by building DNS failure graphs that represent the relations between hosts and failed domains. After the graph is built, a graph decomposition algorithm based on tri-nonnegative matrix factorization is applied to extract dense subgraphs. The authors find that these dense subgraphs are related to various malicious activities. This approach is then applied to 3G mobile network to detect botnets by Br et al. in [26]. In [31] and [30], Choi et al. introduce BotGAD, a system detects botnets by capturing group activities in DNS traffic.

The above techniques require multiple hosts infected by the same botnet existing in the collected traces. Consequently, they have to collect DNS traffic at an upper level (e.g., TLD servers, authoritative servers, etc), or from multiple RDNS servers among networks. The advantage of these works is that evidences from multiple bots can be collected and analyzed. However, they also introduce several challenges. First, the DNS traffic at an upper level is hard to access for most of enterprise and/or university network operators. Second, sharing DNS traffic among networks may introduce privacy issues. Third, it is computationally expensive to run clustering/classification algorithms on large traces collected from multiple networks. Finally, the most significant challenge is that an enterprise network may not have multiple bots, especially belonging to the same botnet. On the contrary, BotDigger can detect an ***individual bot*** by only using DNS traffic collected from a ***single network***.

Schiavoni et al. introduce Phoenix to distinguish DGA domains from non-DGA domains by using both linguistic and IP features [76]. First, Phoenix models pronounceable domains that are generated by human, and assumes DGA domains violate these models. Based on these models, DGA domains are extracted from well-known blacklists. After that, Phoenix groups these extracted domains using domain-to-IP relations. Finally, fingerprints are extracted to label new DGA domains. In [99] and [98], Yadav et al. introduce algorithms to detect DGA domains based on the assumption that current botnets do not use pronounceable language words to avoid conflicting with existing domains. First, the authors use three ways to group DNS queries, for example, all the domains that are mapped to the same IP address are grouped together. Next, the authors calculate several metrics to characterize the distribution of the alphanumeric characters or bigrams of the domains in each group. These metrics include information entropy, Jaccard index and edit-distance. Finally, the above metrics are used to differentiate legitimate domain names from malicious ones. These works can only detect the unpronounceable DGA domains while BotDigger can detect the DGA domains composed of pronounceable words.

In [97], Yadav et al. analyze failed DNS queries around successful DNS queries and use entropy of domains to detect DGA bots. Besides, the authors also introduce a method to speed up the detection of the C&C server IPs. However, the algorithm requires C&C server IP occurring in multiple time windows.

2.6. Traffic Reduction on IDS

Many algorithms reduce load on IDS based on the idea that instead of dropping packets arbitrarily, the packets that are unlikely to contain malicious contents should be discarded.

In [67], Papadogiannakis et al. indicate that the traffic in the early stage of connections are more likely to trigger alerts. In particular, they find that the first 30 packets of the flow trigger 90% of the alerts, and the packets after the 100th packet only trigger 3% of the alerts. Based on these findings, they introduce a technique named selective packet discarding to improve the accuracy of network intrusion detection systems under load. When the IDS is about or already drop some packets, the packets within the cutoff limit are delivered to the IDS and the rest is discarded. A similar idea is used in [28], where the authors only sample a fixed initial set of packets and discard all the remaining traffic. Besides, Maier et al. also indicate that a cutoff of $10-20$ KB per connection is able to retain a complete record of the vast majority of connections based on the heavy-tailed nature of network traffic [62].

In [57] and [56], Limmer et al. improve the IDS performance using dialog-based payload aggregation. The assumption authors make is that most of the signature matches either occur at the beginning of connections or directly after direction changes in data streams. In [55], Limmer et al. introduce Front Payload Aggregation (FPA) technique, in which the first n payload bytes of a flow are added to IPFIX data stream, and then delivered to IDS. This proposed method can work with Vermont [51], a monitoring system that can process Netflow.v9 and IPFIX flow data.

The assumption of the above work is that the first part of a flow is sufficient to raise most alerts. However, the later part of a flow may still include important information. For example, since many browsers use persistent HTTP connections containing multiple HTTP requests [57], some alerts (e.g., a malware binary download, a botnet C&C exchange, etc.) may be missed. To deal with this problem, White et al. introduce a method focusing on "interesting" packets without considering flows [94]. Specifically, they use *Sequential Probability Ratio Test* (SPRT) to classify individual packets as *transparent* if they can be understood by the IDS, and *opaque* otherwise, based on the fact that distribution of byte values in opaque traffic is different from that in transparent traffic. After classification, only transparent packets are delivered to IDS. The downside of this method is that it must examine every packet.

The filtering technique we present is resilient to the above limitations. Our early HE classifiers similarly classify flows as clear and opaque but do not have to examine every packet. Instead, we calculate the entropy of the first M packets in order to classify a flow. Similar to other approaches our filters deliver the first M packets of both clear and opaque flows to IDS, since they may contain protocol related information. However, after the M^{th} packet, if the flow is classified as clear, we continue to deliver the remaining packets to the IDS. Other methods always deliver a fixed number of initial packets to the IDS. Our approach may deliver more packets to the IDS than approaches that always deliver the first few packets, but in return it does not suppress alerts.

Besides reducing the traffic delivered to IDS, researchers also manipulate memory to improve the IDS performance. In [68], Papadogiannakis et al. introduce a two-layer memory management, Selective Packet Paging (SPP) to solve the problems of traffic overloads and algorithmic attacks. A network monitoring framework for stream-oriented traffic processing, Stream capture library (Scap), is presented In [69]. Scap reassembles streams by using *subzero packet copy* technique to minimize data movement, and provides flow-level statistics to user level applications.

CHAPTER 3

Dataset

3.1. Network Traffic

We use seven groups of network traffic datasets in this dissertation. The first group includes 140 Butterfly and Kraken botnet traces obtained from Georgia Tech [34], [75], which we call *140BotTraces*. Each trace in *140BotTraces* was collected by running captured bot binaries in an isolated environment. These traces have roughly similar communication patterns, namely send GET requests to download several executable files, and then GET requests to C&C servers to download C&C data. Finally, the bots start sending spam.

The second group contains botnet traces obtained from Malware Capture Facility Project (MCFP) [10]. The purpose of MCFP is to generate and capture botnet traces in long term. We extracted 41 botnet traces that belong to 17 botnets from MCFP. Each trace was collected by running bot binaries in a sandbox. We name this dataset *MCFPBotTraces* and list the details in Table 3.1.

The third group includes three traces captured at our university lab, which is connected to the outside world via a 10Gb/s link. The lab has a dedicated /24 subnet and runs standard services, such as mail, web, ssh, etc., and also carries traffic for several research projects including two planetlab nodes and a GENI rack. All traces are in tcpdump format and include payload. They are named *Lab1*, *Lab2* and *Lab3*. The first two traces are 24-hour traces captured on Feb-22th-2011 and Dec-11th-2012, respectively. *Lab3* is a 72 hour

Table 3.1. MCFP Botnet Traces

Botnet Name	Number of Traces	Botnet Name	Number of Traces
Zeus	10	Pushdo	1
Neris	4	RBot	2
Donbot	1	Caphaw	1
Conficker	1	Matsnu	1
Cridex	1	Dridex	1
Kazy	4	njRAT	3
Geodo	5	uTorrent	1
Avzhan	1	Kelihos	2
ZeroAccess	2		

trace captured on Dec-13th to Dec-16th-2012. Some statistics for these datasets are given in Table 3.2.

Table 3.2. Lab Traces

	Lab1	Lab2	Lab3
TCP	93.55%	51.87%	72.25%
UDP	1.03%	2.83%	0.258%
ICMP	5.41%	45.14%	27.27%
Others	0.001%	0.149%	0.215%
Bandwidth	14.08 Mb/s	26.16 Mb/s	18.4Mb/s
Peak BW	29.3 Mb/s	133.9 Mb/s	81.9 Mb/s
Duration	24 hours	24 hours	72 hours

The fourth dataset includes flow and entropy information of all incoming and outgoing TCP traffic at our university. This includes more than 30,000 users composed of students and faculty. Due to privacy issues, we do not store payload to disk. Instead we calculate the entropy of each packet in real-time and store it to disk along with flow information. We collected a 24 hour trace on November 19th, 2013. This dataset is named *CSUTrace*.

The fifth dataset includes encrypted traffic only, which we extracted from *Lab1* on HTTPS port 443. Since a successfully established HTTPS connection includes the negotiation of the encryption algorithm and a key exchange, we excluded connections with less than 5 packets with payload. This dataset is named *Lab1-HTTPS*.

The sixth dataset includes DNS traffic captured from our university. For all the users connected to the university network, their DNS queries are sent to four recursive DNS servers. We collect all the DNS traffic by mirroring the ports to the four servers for a total of seven days, starting from April 2nd to 8th in 2015. We call this dataset *CSUDNSTrace*. *CSUDNSTrace* includes 1.15E9 domains queried by 20682 university hosts. 5.5E6 of the queried domains are unique.

The seventh dataset includes domains generated by a well known DGA botnet - Conficker. Variant C of Conficker generates 50,000 domains every day [53]. We collected a list of all the 1,500,000 domains generated in April 2009 from tools provided by Institute of Computer Science at University of Bonn [52].

3.2. Static Application Content

We use two groups of static application content in our experiments. The first comprises of 11 types of non-encrypted files, namely txt, MS word, excel, pdf, mp3, jpeg, portable executable (exe), gzip, bzip, rar and winzip. We use a total of 1000 files per type. The mp3 and txt files were downloaded from *archive.org*. Exe files were collected from a website providing free software, *www.skycn.com*. Jpeg files were obtained from the official websites of *National Geographic* and *NASA*. We collected MS word and excel files from *Google* by performing the file type search (e.g., filetype:docx). Pdf files were picked from academic conferences and journals. We randomly selected 330 MS word, 330 excel and 340 txt files from the above pool, and then applied gzip, bzip, rar and winzip to generate the compressed files. This group of files is named *NonEncryptedFiles*.

The second group contains encrypted files. We use the same 1000 files we used to generate compressed files above, and feed them to a commercial data encryption software, named *Advanced Encryption Package Pro* (AEP) [1]. We encrypt these 1000 files using 17 encryption algorithms provided by AEP. The encryption algorithms are listed in Table 4.2. This group of files is named *EncryptedFiles*.

TABLE 3.3. Encryption Algorithms

Encryption Algorithm	Key Length (bit)	Encryption Algorithm	Key Length (bit)	Encryption Algorithm	Key Length (bit)
DESX	128	Blowfish	448	AES	256
CAST	256	Triple-DES	192	RC2	1024
Diamond 2	2048	Tea	128	Safer	128
3-Way	96	GOST	256	Shark	128
Square	128	Skipjack	80	Twofish	256
MARS	448	Serpent	256		

CHAPTER 4

BotTalker: Generating Encrypted, Customizable C&C Traces

Encrypted botnet traffic makes detection much harder because many detection systems rely on DPI techniques. In addition, encryption makes it very hard to get ground truth necessary to develop detection algorithms. While bot communication can be captured in sandboxes from binaries harvested around the net, traces collected from sandboxes are fairly static. Researchers would like to adjust parameters such as encryption type, timing, etc., the same way a bot might act to alter its signature and foil detection.

To address these needs we designed and implemented BotTalker: a tool that researchers can use to generate customized encrypted botnet traffic. BotTalker emulates the actions a bot would take to encrypt communication and produces traces that look like they come from real botnets. BotTalker includes a highly configurable encrypted-traffic converter along with real, non-encrypted bot traces and background traffic. The converter is able to convert non-encrypted botnet traces into encrypted ones. It enables customization along three dimensions: (a) selection of real encryption algorithm, (b) flow or packet level conversion, SSL emulation and (c) IP address substitution. Many standard encryption algorithms are supported, including AES, DES, RC4 and more. The reason for providing packet/flow conversion and SSL emulation is to support a broad range of encryption methods. In packet conversion we emulate the case where a bot encrypts packets individually as they are transmitted. Flow conversion emulates the case where a bot transmits encrypted objects. SSL conversion emulates the case when the botnet exchange information via SSL connections. Finally, our converter provides the ability to obfuscate IP addresses. This is needed because many bot detection tools look for communication with blacklisted IP addresses, which can easily be subverted. To our best knowledge, BotTalker is the first work that provides user customized encrypted botnet traffic converted from real botnet traffic, rather than simulated traffic.

4.1. BotTalker Design

To achieve the goal of emulating the actions a bot takes to encrypt communications, we first introduce several requirements that BotTalker needs to satisfy. Then we describe the architecture.

4.1.1. Requirements.

(1) Retain network level characteristics of original traces:

Network level characteristics include timestamps, packet size, flow size, etc. These can be typically calculated from traffic at the network level. Retaining these characteristics is important because many IDSs examine them to detect malicious behaviors.

(2) Use of real encryption algorithms:

Many real botnets use standard encryption algorithms because they are easy to use. For example, Rustock, Storm and Waledac use RC4, XOR and AES to encrypt communications respectively. BotTalker supports many standard encryption algorithms to encrypt the communication payload.

(3) Emulate various encryption schemes:

When a bot encrypts traffic it may encrypt one packet at a time, the entire flow, simply use an existing encrypted channel such as SSL. Packet level encryption emulates the case where packets are encrypted individually. For example, when the controller transmits a stream of commands to the bots via the same connection, it may put each command into a single packet, and each packet is encrypted individually. Flow level encryption emulates the case where an entire object is transmitted, such as binaries, spam lists, etc. When a bot wants to transfer these encrypted objects, it usually encrypts them in memory first, then the entire encrypted data is chopped into multiple packets and sent out. SSL can be used when exchanging information via HTTPS, which is typically not blocked at firewalls. In fact, Bayer found that many malware already use SSL to protect their communications [24]. BotTalker provides the capability to emulate a botnet's communication via SSL.

(4) Use real background traffic:

The use of real background traffic and real bot traffic is important to assess false positives and false negatives of detection algorithms. However, blending bot traffic and background traffic when such traces were collected in different networks has many challenges. For example, does one pick existing hosts from the background traffic and assign bot traffic to them? Or do we introduce new hosts along with those present in the background traffic trace? BotTalker supports both approaches.

4.1.2. BotTalker Architecture. We design BotTalker to satisfy the above requirements. The architecture is shown in Figure 4.1. BotTalker includes a highly configurable encrypted-traffic converter along with background traffic and real, non-encrypted bot traces. The converter is the core module. There are two inputs to the converter: the bot trace to encrypt and a set of parameters (e.g., encryption algorithm, packet or flow level conversion). After generating the encrypted botnet traces, users may optionally blend it with the provided background traffic or with background traffic provided by the user. The final result is now ready to use.

BotTalker comes with a collection of bot traffic and background traffic. At the moment, it provides 140 Butterfly and Kraken bot traces [34], [75], as well as traffic from the 2009 DARPA dataset. Note that it is trivial to add more traffic to the current collection.

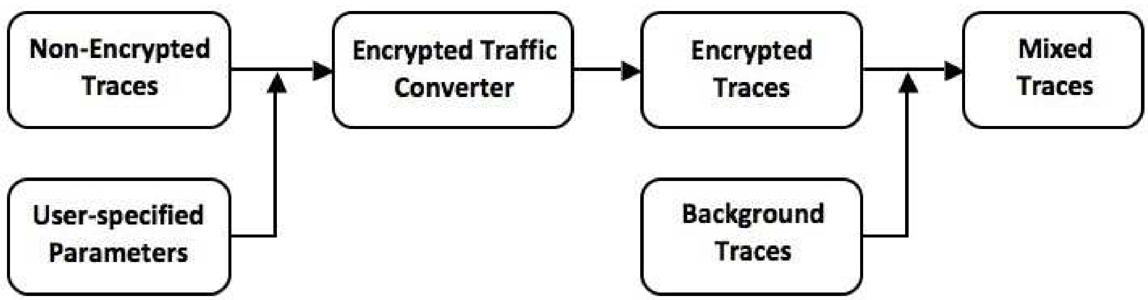

FIGURE 4.1. BotTalker Architecture

4.2. Encryption Algorithms

Stream and block ciphers are two main approaches in encrypting data. Stream ciphers encrypt plaintext of variable length. Block ciphers encrypt fixed-length groups of bits called blocks. In our implementation, we support both stream and block ciphers. In particular, four modes of block ciphers are supported, including Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Output Feedback (OFB). Among these modes, ECB and CBC require the data size be a multiple of block size, while CFB and OFB do not. The various standard encryption algorithms provided by BotTalker are shown in Table 4.1. We implement these encryption algorithms based on OpenSSL[12].

Table 4.1. BotTalker Encryption Algorithms

Encryption Algorithm	Cipher Mode
XOR	N/A
RC4	N/A
RC4_40	N/A
DES	ECB. CBC, CFB, OFB
2-key Triple-DES	ECB CBC, CFB, OFB
3-key Triple-DES	ECB, CBC, CFB, OFB
DESX	CBC
RC2	ECB, CBC, CFB, OFB
Blowfish	ECB, CBC, CFB, OFB
CAST	ECB, CBC, CFB, OFB
AES_128	ECB, CBC, CFB1, CFB8, CFB128, OFB
AES_192	ECB, CBC, CFB1, CFB8, CFB128, OFB
AES_256	ECB, CBC, CFB1, CFB8, CFB128, OFB

4.3. Supported Encryption Schemes

In this section, we describe three encryption schemes that BotTalker supports. These are packet level encryption, flow level encryption and SSL.

4.3.1. Packet Level Encryption.

The basic mechanism of packet level encryption is shown in Figure 4.2. There are three steps in packet level encryption.

(1) Extract the payload from each individual packet.
(2) Encrypt the payload with the user specified encryption algorithm.
(3) Place the encrypted payload back in the packet.

This approach essentially leaves the header intact and changes only the payload. One challenge, however, is that the size of the payload, once encrypted, could be different than the original payload. For example, with a stream cipher (XOR) or CFB and OFB modes of block ciphers the size of the payload remains the same. But with block ciphers using ECB and CBC the size of encrypted data is larger than the original due to padding. If we simply put the encrypted payload back into the packet the size of packet will change, which means that the sequence and acknowledgment numbers for all subsequent packets need to change. Moreover, if the size of the encrypted packet is greater than the MTU an additional packet has to be generated, and the timestamps of all subsequent packets have to be adjusted. To avoid this complicated scenario we compromise by simply trimming the encrypted payload to the same size as the original packet. We acknowledge that the encrypted stream is now different than what a bot may have produced, but we believe that it is more important to preserve the other characteristics of the stream (timing, header information, etc.) than strive for a completely accurate representation of the encrypted bytes. The results described later in the paper indicate that this is not a serious problem.

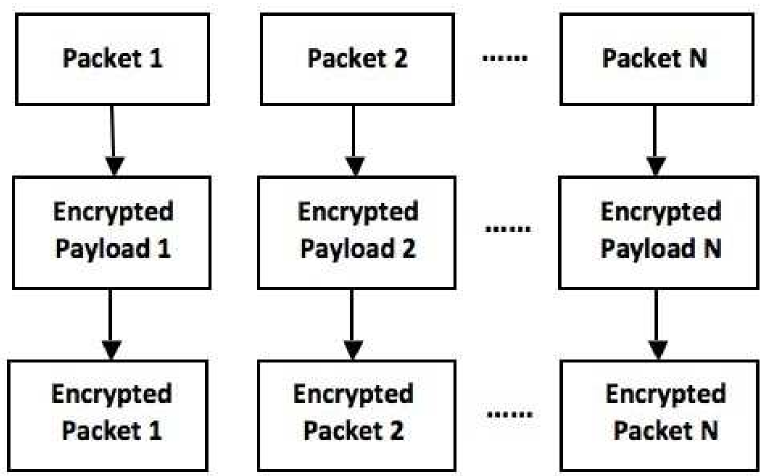

FIGURE 4.2. Packet Level Encryption

4.3.2. FLOW LEVEL ENCRYPTION. Figure 4.3 depicts the process of flow level encryption. With flow encryption we assume that the endpoints exchange symmetric keys and encrypt each direction with a different key, but the same encryption algorithm. We also detect spurts of packets and encrypt them as a group so if the encrypted version is larger we do not alter packet timing. Flow level encryption is performed as follows:

(1) Extract and concatenate the payload from groups of packets in each direction.

(2) Encrypt each group or spurt using the user's specified encryption algorithm.

(3) Place the encrypted payload back into the original packets.

With flow encryption we face the same problem as packet encryption scheme, namely the possibly larger size of encrypted data. We use the same solution: we trim the encrypted data to the same size as the original before we place it back into packets. In the flow case, we only trim the last packet in a spurt.

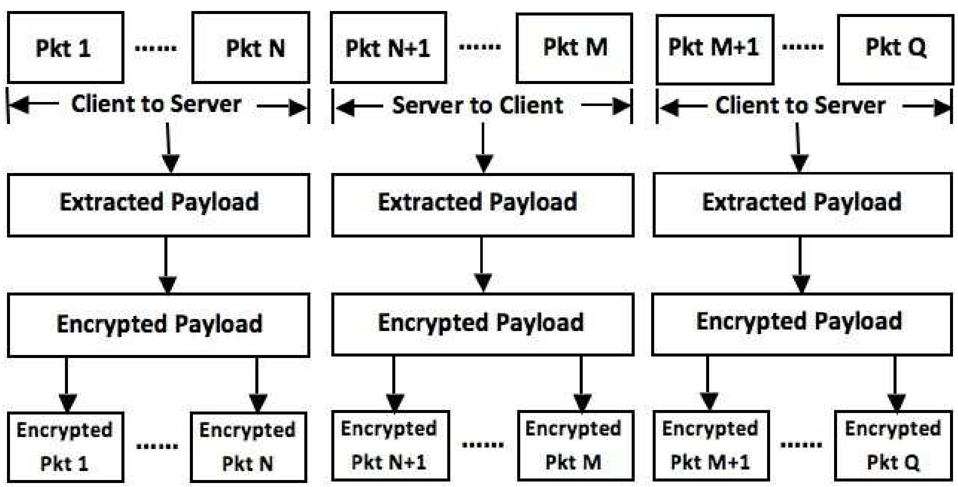

FIGURE 4.3. Flow Level Encryption

4.3.3. SSL EMULATION. The process of SSL emulation is shown in Figure 4.4. The figure indicates that the first step is similar with flow level encryption, which is to group packets in spurts on each direction, extract and concatenate the spurt's payload. Then we establish a SSL connection, transmit the extracted payloads in their original direction and capture the resulting packet trace. The IP addresses, port numbers and timestamps of the captured SSL traffic are different than the original. While it is easy to replace the IP addresses and ports, a big challenge is to adjust the timestamps in the SSL trace because the total number of packets is different. For example, the trace includes packets for setting up the SSL state.

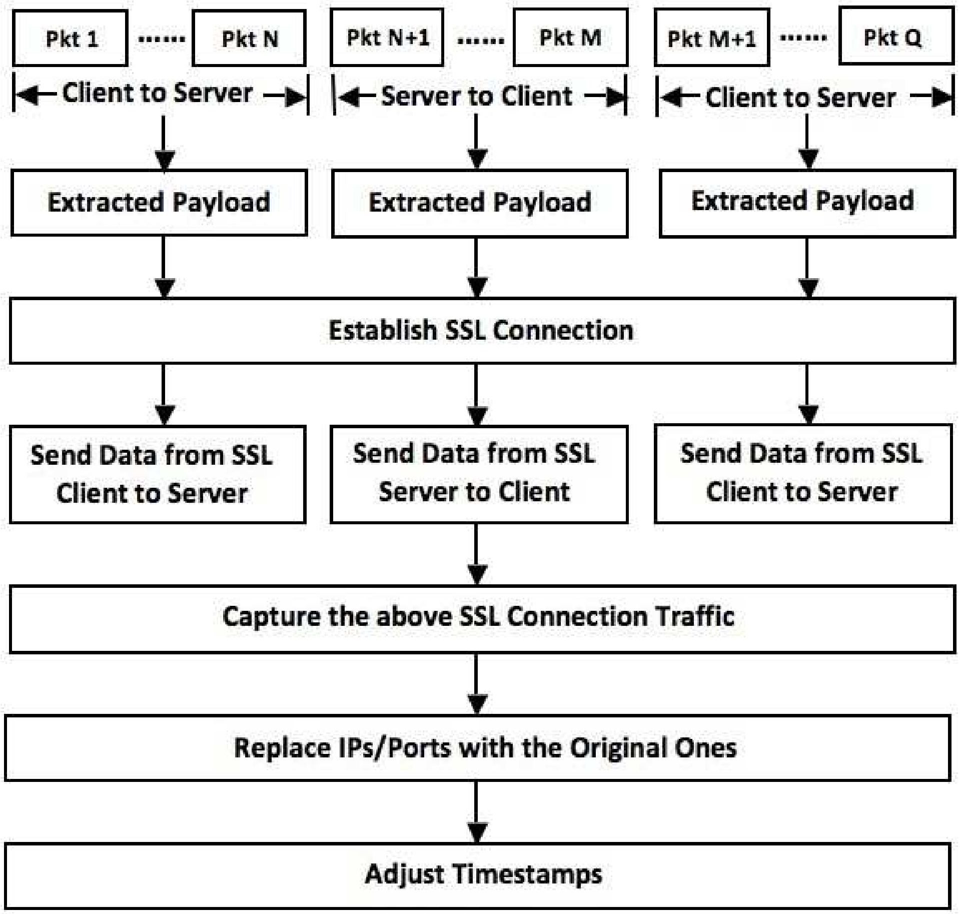

FIGURE 4.4. SSL Emulation

We deal with this issue by calculating the *Client Reaction Time* (T_1), *Packet Inter-arrival Time on Server* (T_2), *Round Trip Time* (T_3) and *Packet Inter-arrival Time on Client* (T_4) from the original trace, as shown in Figure 4.5. Then we assign the timestamp of the first packet in the original trace to the first packet in the SSL trace. Finally, we use the above four values to estimate the timestamps for subsequent packets. For example, if the first packet is sent from client to server and the second is sent from server to client, then *Time(2nd pkt)* =

Time(1st pkt) + T_3. We believe that this is a reasonably accurate representation of a SSL connection.

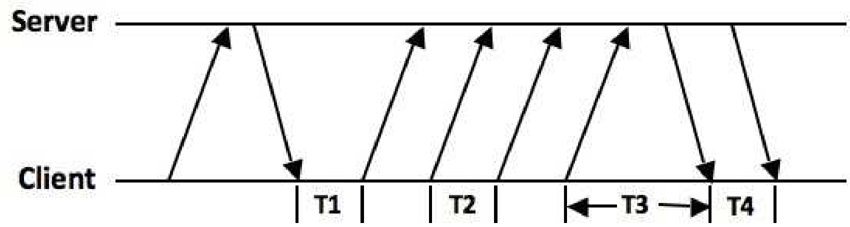

FIGURE 4.5. SSL Timestamps Emulation

4.4. DISCUSSION

4.4.1. DO WE NEED SO MANY ENCRYPTION SCHEMES? PROVEX [74] is a botnet detection system that can detect encrypted C&C channels. It relies on the fact that many encrypted botnets use static keys, or their keys can be extracted from the transmitted data. With the help of sandboxes and reverse engineering, PROVEX is able to collect some botnet binaries and obtain a set of keys or key extraction functions from them. After that, PROVEX can detect encrypted C&C channels by i) extracting C&C message patterns from captured botnet binaries, ii) decrypting every packet using the set of keys or key extraction functions, and iii) matching the decrypted message with the extracted C&C patterns. However, PROVEX can only decrypt individual packets and does not provide stream reassembly mechanisms. As mentioned earlier, when bots transfer a binary they encrypt the entire file first and then chop the encrypted data into packets. In this case, PROVEX may not work because some encryption algorithms require reassembling entire encrypted data to get full decrypted messages. However, if the botnet uses simple encryption algorithms (e.g. XOR), the first decrypted packet still has the correct original message and PROVEX can make use of it. This example shows the need to generate encrypted data sets using different encryption schemes with different encryption algorithms to test the detection systems' capabilities.

4.4.2. Do We Need So Many Encryption Algorithms? Botzilla [73] can detect malware when the latter contacts C&C servers. Botzilla assumes that in the communication with the servers, content may vary if they rely on host and network settings (e.g., OS, current time), but others do not change and can be used as signatures. Based on this assumption, Botzilla repeatedly runs the binaries in a controlled environment by changing network and host settings, and extracts the invariant content patterns from the traffic as signatures. Botzilla does not have any ability to decrypt encrypted messages and is not designed to detect encrypted communications. However, when the authors test it on Storm Worm, Botzilla is able to detect the communication even though it is encrypted. The reason is that Storm Worm uses a simple encryption algorithm (XOR) and does not conceal communication very well. If a strong encryption algorithm were used by the bots, Botzilla would most likely not detect it.

BotTalker's rich set of encryption algorithms allows us to evaluate PROVEX's performance. As PROVEX cannot distinguish which encryption algorithm and key is used in captured packets, it has to try all encryption algorithms and keys at its disposal. To test its performance we can generate a set of encrypted botnet traces using various encryption algorithms and keys, and then feed them to PROVEX.

4.5. Summary

Datasets with ground truth play an important role in developing detection systems. However, collecting such data sets is very hard. We address this issue by introducing BotTalker, a tool that can be used to generate customized encrypted botnet communication traffic.

BotTalker works by emulating the actions a real bot would take to encrypt communication. BotTalker retains network level characteristics of original botnet traces, uses real encryption algorithms, and emulate various encryption schemes. The encrypted botnet traces generated by BotTalker benefits IDS developers, IDS customers and also IDS benchmark.

CHAPTER 5

Evaluating the Damage Result from Encryption on Security Systems

In this chapter, we use BotTalker to generate encrypted botnet traffic and evaluate the damage result from encrypted botnet traffic on a widely used botnet detection system - BotHunter [41] and two IDSs, Snort [13] and Suricata [14]. The results show that the encrypted botnet traffic foils bot detection in these systems.

5.1. Generating Encrypted Botnet Traces

Armed with BotTalker, we generate 6 datasets by applying encryption and/or IP replacement using the non-encrypted botnet datasets 140BotTraces and MCFPBotTraces. Botnets can easily change C&C servers to evade blacklists, so we replace all IP addresses to addresses that do not appear on blacklists. We name these two datasets as *140ReplaceIPsBots* and *MCFPReplaceIPsBots*. Then, we generate two datasets to emulate botnets using encrypted communication. For each trace in 140BotTraces and MCFPBotTraces, we encrypt all the traffic except SMTP and DNS by applying RC4 encryption with packet level encryption scheme. We call these data sets *140EncryptedBots* and *MCFPEncryptedBots*, respectively. Finally, we apply IP replacement on the above encrypted datasets to obtain new datasets, namely *140EncRepIPsBots* and *MCFPEncRepIPsBots*.

5.2. Reverse Engineering BotHunter

BotHunter (BH) is a real-time bot detection system. The fundamental observation in BH is that bot infection follows a distinct set of events in some loose order. BH includes eight infection events, such as *Inbound Scan*, *Inbound Infection*, *Egg Download*, *C&C communication*, *Outbound Scan*, etc [4]. BH builds a detection system that scores hosts engaging in such activity and flags them as bots if the score exceeds a threshold (0.8 in the current implementation).

BH makes use of Snort. It introduces specific Snort rules where appropriate, to detect events of interest and then uses the resulting Snort output to aid in bot detection. Some of the Snort rules apply DPI to search for specific patterns, such as magic numbers to detect executable downloads or signature strings associated with known bots.

As mentioned earlier, each event is associated with a score. Since the source code for BH is not available to us, it is hard to determine these scores directly. To determine the individual event scores, we apply reverse engineering techniques as follows: we disable the Snort rules that correspond to specific events, and then measure the difference in the final score between host profiles associated with the disabled event. For example, if the original score without disabling any Snort rules is 3.5 and the score after disabling the rules to detect *Egg Download* is 3.0, then we conclude that *Egg Download* has a score of 0.5. Our bot traces trigger six out of eight events in BH and the scores are shown in Table 5.1.

5.3. Damage Result from Encryption on BotHunter

We first run BH on the bot traces 140BotTraces, 140ReplaceIPsBots, 140EncryptedBots and 140EncRepIPsBots. Table 5.2 shows the results. From the table we observe that 138 out of 140 bots are detected on the non-encrypted botnet traces. When BH detects a bot

TABLE 5.1. BotHunter Scores

Event Alert	Score
Egg Download	0.5
C&C Communication based on DPI	0.5
C&C Communication based on RBN	0.1
Outbound Attack	0.5
Outbound Scan based on Behavior	0.3
Attack Preparation	0.5
Bot Declaration	0.8

it also reports the rules that fired. We categorize these rules into four groups: 1) DPI on HTTP traffic: these rules triggered 2 events with overall score of 1.0. 2) SMTP traffic: these rules triggered 1 event whose score is 0.5. 3) DNS traffic: these rules triggered 2 events with overall score of 1.3. 4) Blacklists: these rules triggered 1 event whose score is 0.1.

When we feed 140ReplaceIPsBots to BH, 138 bots are detected as they trigger the events in groups 1 and 2 (some also trigger events in group 3). With encryption only, 127 bots are missed because they only trigger events in group 2 and 4 whose score is less than the threshold. The other 13 bots are detected because they trigger events in group 2, 3 and 4. These 13 bots are still detected when both encryption and IP replacement are applied as they trigger events in group 2 and 3.

TABLE 5.2. BotHunter Detection Rate on 140 Samples Botnet Traces

Dataset	Traffic Encryption	IP Replacement	Detected Bots
140BotTraces	Not Applied	Not Applied	138
140ReplaceIPsBots	Not Applied	Applied	138
140EncryptedBots	Applied	Not Applied	13
140EncRepIPsBots	Applied	Applied	13

We then run BH on MCFP bot traces, including MCFPBotTraces, MCFPReplaceIPsBots, MCFPEncryptedBots and MCFPEncRepIPsBots. The results are listed in Table 5.3. From the table we can see that BH only detects 5 out of 41 bots even no encryption or IP replacement is applied. Besides the 5 detected bots, BH also generates alerts for another 13 bots. However, these 13 bots are not labeled because the scores of triggered alerts are less than the threshold. One explanation for such low bot detection rate is that the bot traces may not contain enough malicious traffic. For example, the bots may detect they are running on virtual machines and stop sending C&C messages or performing attacks. Note that we are not using these botnet traces to evaluate how well BH detects bots. Instead, we use these traces to evaluate the damage result from encryption on BH. From Table 5.3 we can also find that when we feed MCFPReplaceIPsBots and MCFPEncryptedBots to BH, 5 bots are still detected. The reason is that many rules are triggered and they relay on DPI, blacklist and also SMTP. As a result, even encryption or IP replacement evasion technique makes the DPI or blacklist rules blind, the score of the other triggered rules is still above the threshold thus BH detects the bots. When both encryption and IP replacement are applied, BH fails to detect one of the above 5 bots because all the rules used to detect it rely on DPI and blacklist, which are now foiled.

Table 5.3. BotHunter Detection Rate on MCFP Botnet Traces

Dataset	Traffic Encryption	IP Replacement	Detected Bots
MCFPBotTraces	Not Applied	Not Applied	5
MCFPReplaceIPsBots	Not Applied	Applied	5
MCFPEncryptedBots	Applied	Not Applied	5
MCFPEncRepIPsBots	Applied	Applied	4

Now we draw two conclusions for our findings. First, DPI technique plays a more important role than blacklist in BH. Second, encrypted botnet traffic largely foils bot detection in BH.

5.4. Damage Result from Encryption on Snort

In this section, we measure the damage result from encrypted bot communication on a widely used IDS - Snort (version 2.9.6.2) using the Emerging Threats Open Ruleset [6]. The ETOpen Ruleset is an anti-malware IDS/IPS ruleset that helps users to enhance their network-based malware detection. We use two criteria to measure the damage. One is the number of triggered alerts, and the other is the number of detected bots.

First we feed non-encrypted 140BotTraces to Snort. 14982 alerts are triggered. The number of triggered alerts for each bot is shown in Figure 5.1. The x-axis is the bot index and the y-axis is the number of alerts. Although most of the alerts in Figure 5.1 are related to SMTP, they are triggered by the same rule. So we consider unique alerts in Figure 5.2. From Figure 5.2 we see that most of the unique alerts are DPI related.

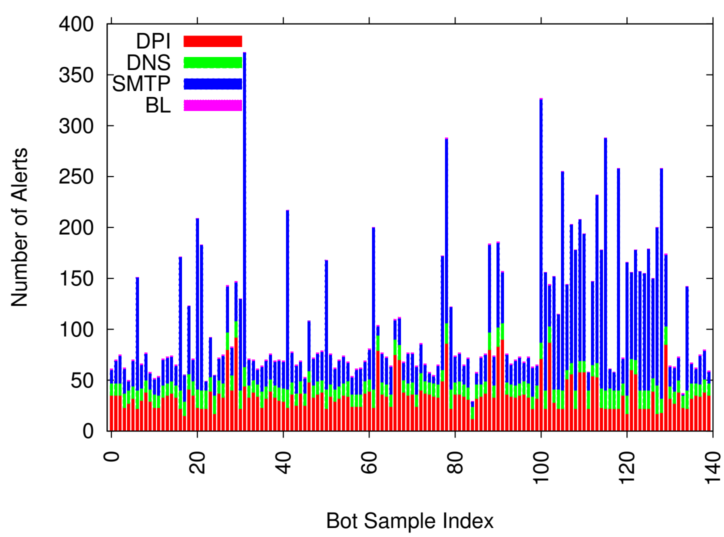

Figure 5.1. Snort Alerts per Bot without Encryption (140BotTraces)

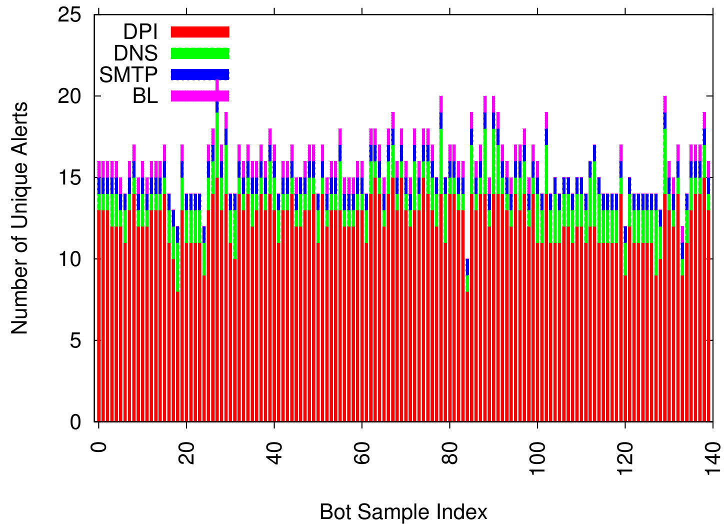

FIGURE 5.2. Snort Unique Alerts per Bot without Encryption (140BotTraces)

We then feed the encrypted 140EncryptedBots to Snort. We plot the triggered alerts and unique alerts in Figure 5.3 and Figure 5.4, respectively. From the figures we can see that only the alerts related to DNS, SMTP and blacklist are triggered, but all the DPI related alerts disappear. We compare the alerts of using non-encrypted and encrypted botnet datasets in Table 5.4. From the table we can see that when encryption is applied, Snort misses 33.8% of total alerts and 78.7% of unique alerts.

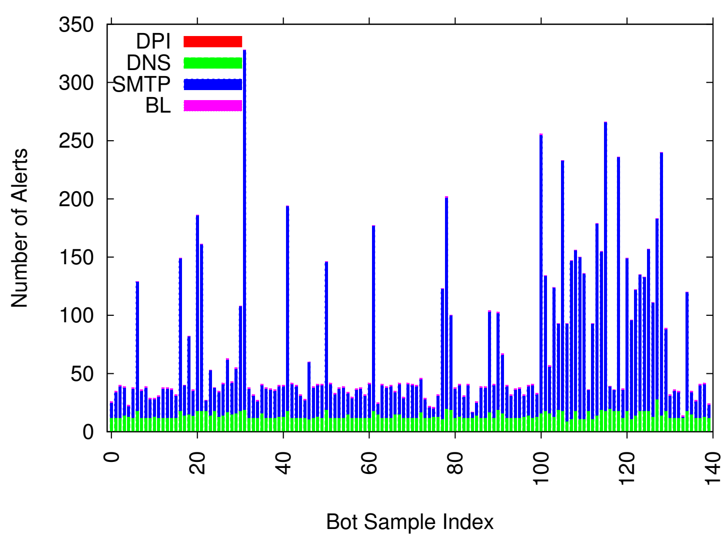

FIGURE 5.3. Snort Alerts per Bot with Encryption (140EncryptedBots)

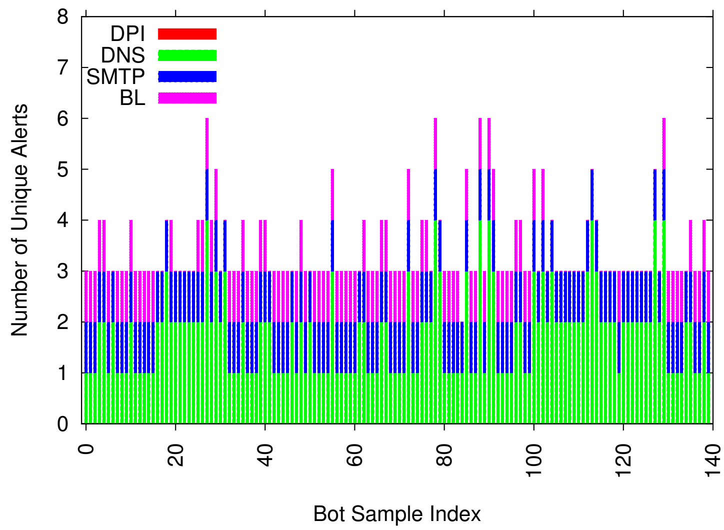

FIGURE 5.4. Snort Unique Alerts per Bot with Encryption (140EncryptedBots)

TABLE 5.4. Snort Alerts of 140BotTraces and 140EncryptedBots

	Total Alerts	Unique Alerts
140BotSamples	14982	2223
140EncryptedBots	9926	478
Encryption Damage	5056	1745
Damage Percentage	33.8%	78.7%

Now we run Snort on MCFP bot traces, including MCFPBotTraces and MCFPEncryptedBots, to repeat the above experiments. 562235 alerts in total and 317 unique alerts are triggered when we feed MCFPBotTraces to Snort. The total alerts and unique alerts are plotted in Figure 5.5 and Figure 5.6 (alerts for a specific bot is not included in the figures because it triggers too many alerts to show in the figures). Then we feed MCFPEncryptedBots to Snort. We plot the triggered alerts and unique alerts in Figure 5.7 and Figure 5.8. From the figures we can see that most of the DPI-related alerts disappear, and only 318935 alerts and 89 unique alerts are triggered. However, 10 unique DPI-related alerts are occasionally triggered because the encrypted traces match some DPI rules that being used to detect Edonkey traffic. We compare the alerts of using non-encrypted and encrypted botnet datasets in Table 5.5. From the table we can see that when encryption is applied, Snort misses 43.3% of total alerts and 71.9% unique alerts.

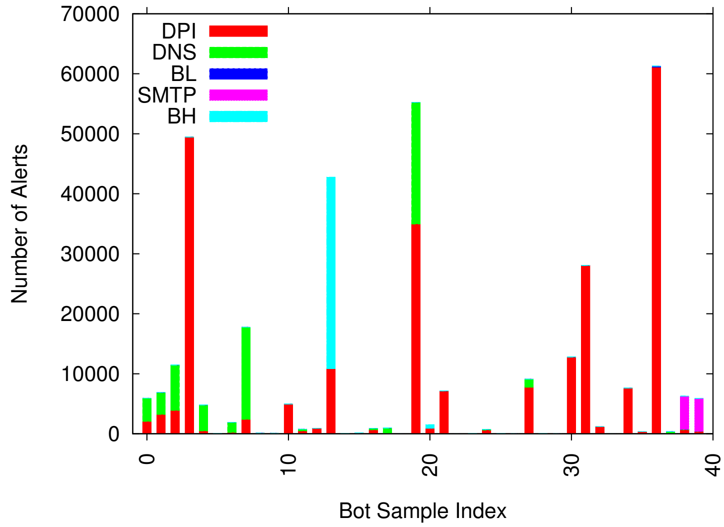

FIGURE 5.5. Snort Alerts per Bot without Encryption (MCFPBotTraces)

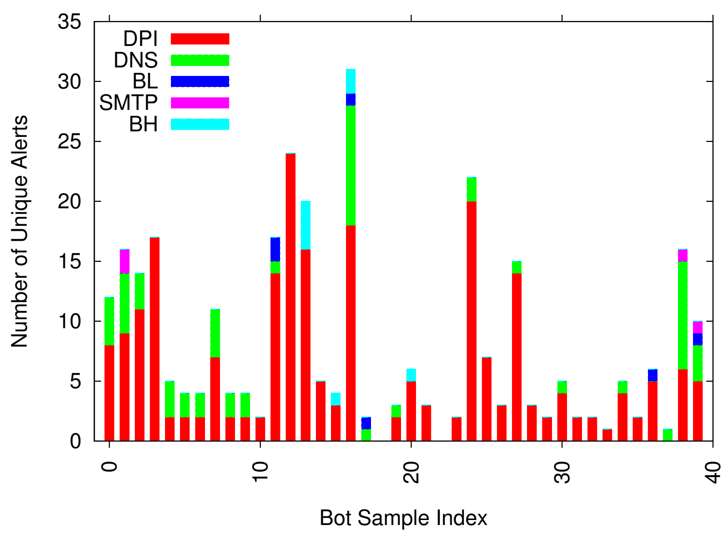

FIGURE 5.6. Snort Unique Alerts per Bot without Encryption (MCFPBotTraces)

Now we use another criterion - the number of detected bots, to evaluate the damage result from encrypted botnet traffic on Snort. In the above experiments, we have seen that Snort generates lots of alerts on the non-encrypted as well as the encrypted botnet traces. Upon further investigation, we find that some triggered alerts are strongly related to bot

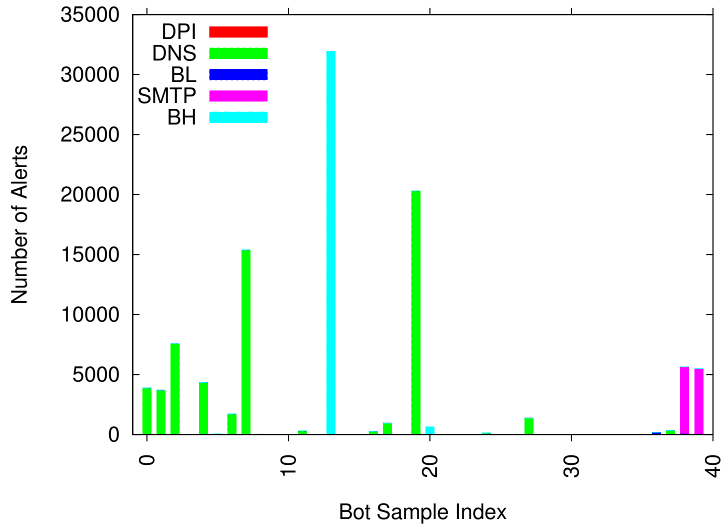

FIGURE 5.7. Snort Alerts per Bot with Encryption (MCFPEncryptedBots)

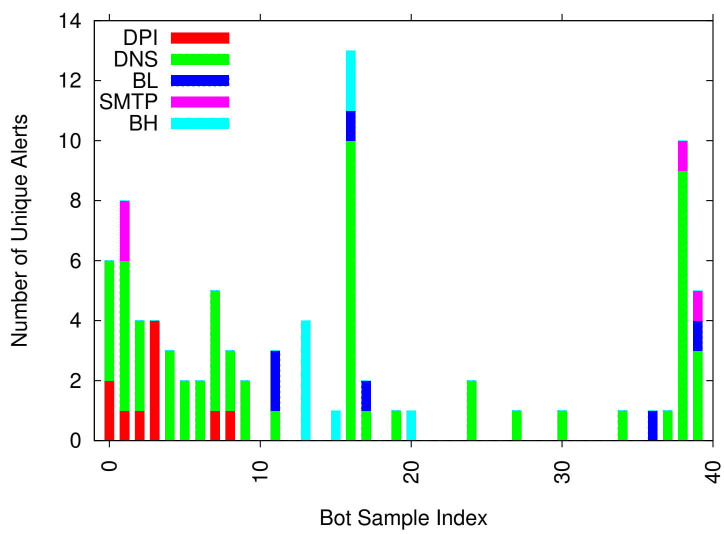

FIGURE 5.8. Snort Unique Alerts per Bot with Encryption (MCFPEncryptedBots)

TABLE 5.5. Snort Alerts of MCFPBotTraces and MCFPEncryptedBots

	Total Alerts	Unique Alerts
MCFPBotTraces	562235	317
MCFPEncryptedBots	318935	89
Encryption Damage	243300	228
Damage Percentage	43.3%	71.9%

infection. Based on these alerts, we use a simple method to label bots. For a given host, if any of its triggered alerts indicates bot infection, we consider the host as a bot. It is true that this simple method may introduce false positives because a legitimate host may trigger such rules occasionally. However, note that our aim is to measure the damage result from encryption instead of developing a bot detection system.

First we measure the damage using the 140 bot traces. When we use the non-encrypted 140BotTraces as input, all of 140 bots are labeled as bots. However, only 95 out of the 140 bots are detected when 140EncryptedBots are fed into Snort, meaning that 45 (32.1%) bots are missed due to encryption. Then we use the MCFP bot traces to measure the damage. When MCFPBotTraces and MCFPEncryptedBots are used as inputs to Snort, 27 and 12 bots are detected, indicating that 15 (55.6%) bots are missed due to encryption.

Now we conclude the damage result from encryption on Snort. In summary, encryption greatly foils Snort. For our two particular datasets, encrypted botnet communications 1) suppress more than 33% of total alerts and more than 70% of unique alerts for Snort, 2) make Snort miss 32% and 55% of bots.

5.5. Damage Result from Encryption on Suricata

In this section, we measure the damage result from encrypted bot communication on Suricata (version 2.0.4). The experiments are very similar as the last section. First we feed non-encrypted 140BotTraces to Suricata, 13453 alerts and 2339 unique alerts are triggered. The number of total alerts and unique alerts are plotted in Figure 5.9 and Figure 5.10. From the figure we can see that most of the unique alerts are DPI-related. We then feed the encrypted 140EncryptedBots to Suricata. We plot the number of total alerts and unique alerts in Figure 5.11 and Figure 5.12. The figures indicate that only the alerts related to

DNS, SMTP and blacklist are triggered. We compare the alerts of using non-encrypted and encrypted botnet traces in Table 5.6. From the table we can find that when encryption is applied, Suricata misses 36.1% of total alerts and 75.2% of unique alerts.

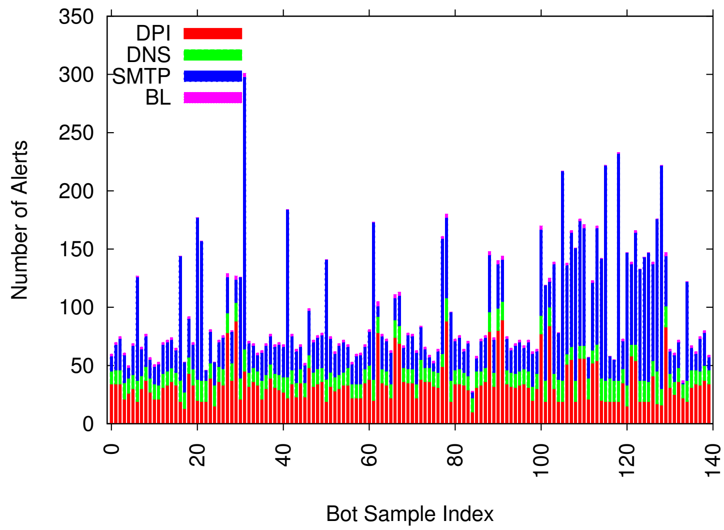

FIGURE 5.9. Suricata Alerts per Bot without Encryption (140BotTraces)

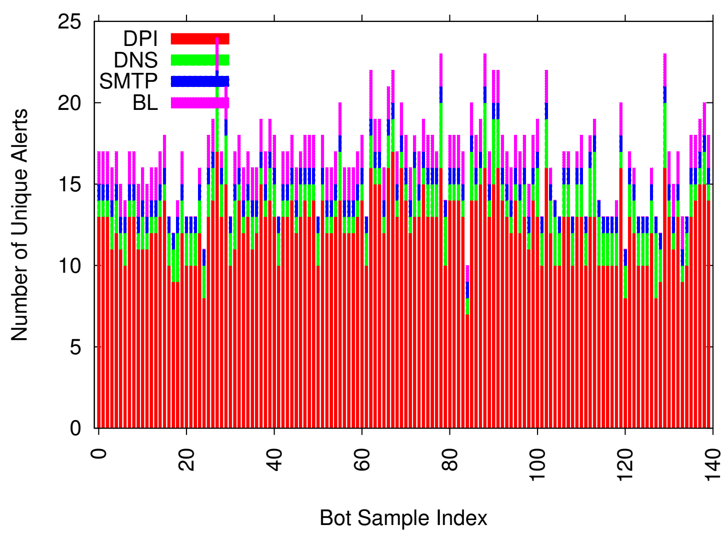

FIGURE 5.10. Suricata Unique Alerts per Bot without Encryption (140BotTraces)

Then, we use MCFP datasets MCFPBotTraces and MCFPEncryptedBots as inputs to Suricata. 570155 alerts and 291 unique alerts are triggered when we feed MCFPBotTraces to Suricata. The number of total alerts and unique alerts are plotted in Figure 5.13 and

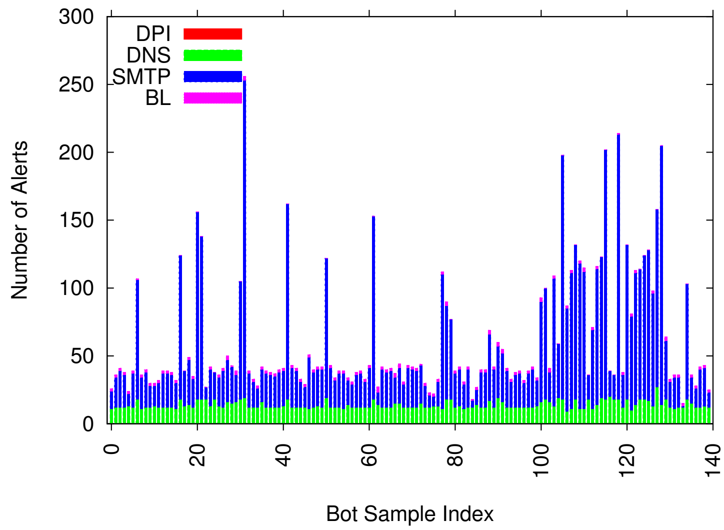

FIGURE 5.11. Suricata Alerts per Bot with Encryption (140EncryptedBots)

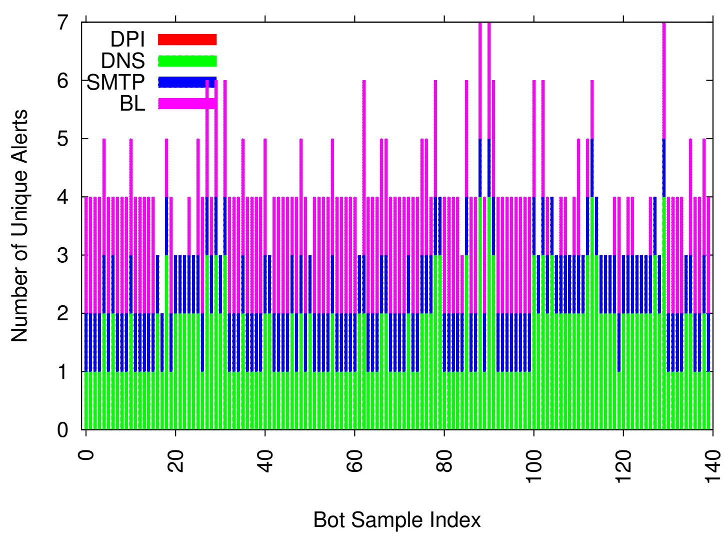

FIGURE 5.12. Suricata Unique Alerts per Bot with Encryption (140EncryptedBots)

Figure 5.14. Then we use MCFPEncryptedBots as input to Suricata. We plot the number of total alerts and unique alerts in Figure 5.15 and Figure 5.16. From the figures we can see that most of the Suricata DPI-related alerts disappear. We compare the alerts of using non-encrypted and encrypted botnet traces in Table 5.7. The table tells us that when encryption is applied, Suricata misses 46.2% of total alerts and 74.2% of unique alerts.

TABLE 5.6. Suricata Alerts of 140BotTraces and 140EncryptedBots

	Total Alerts	Unique Alerts
140BotSamples	13453	2339
140EncryptedBots	8591	580
Encryption Damage	4862	1759
Damage Percentage	36.1%	75.2%

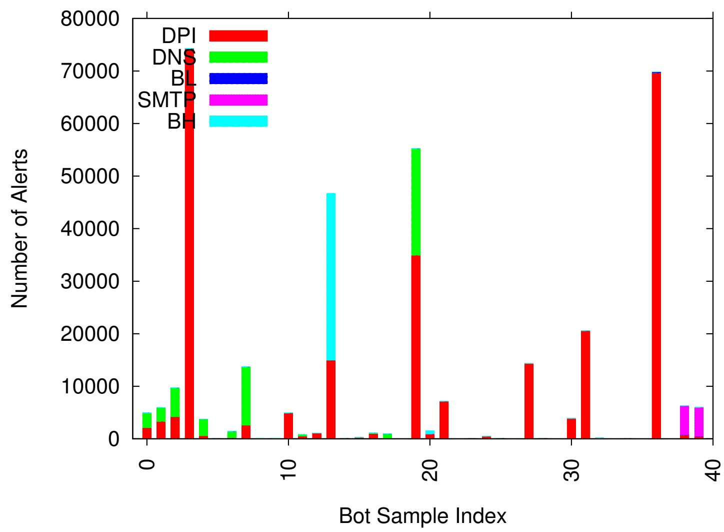

FIGURE 5.13. Suricata Alerts per Bot without Encryption (MCFPBotTraces)

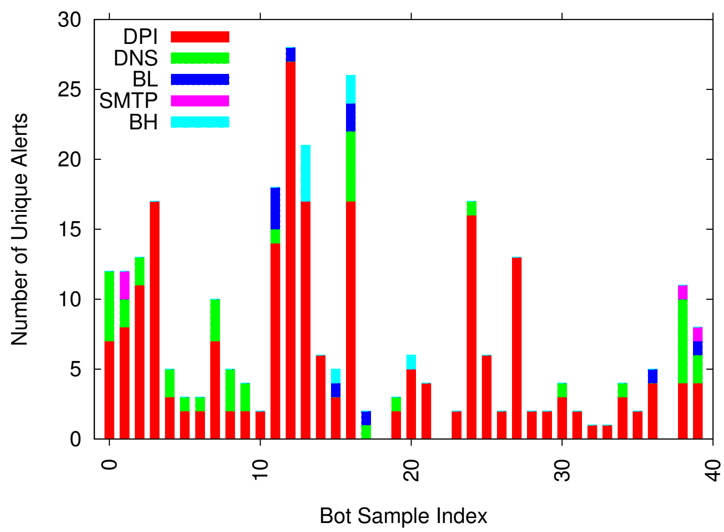

FIGURE 5.14. Suricata Unique Alerts per Bot without Encryption (MCFP-BotTraces)

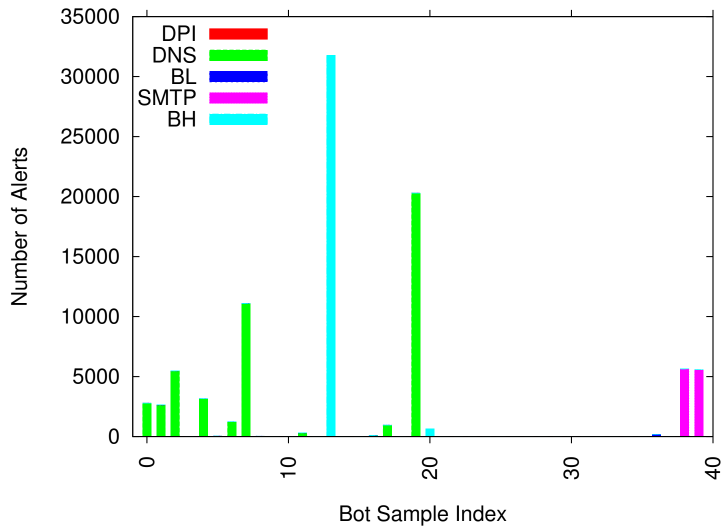

FIGURE 5.15. Suricata Alerts per Bot with Encryption (MCFPEncryptedBots)

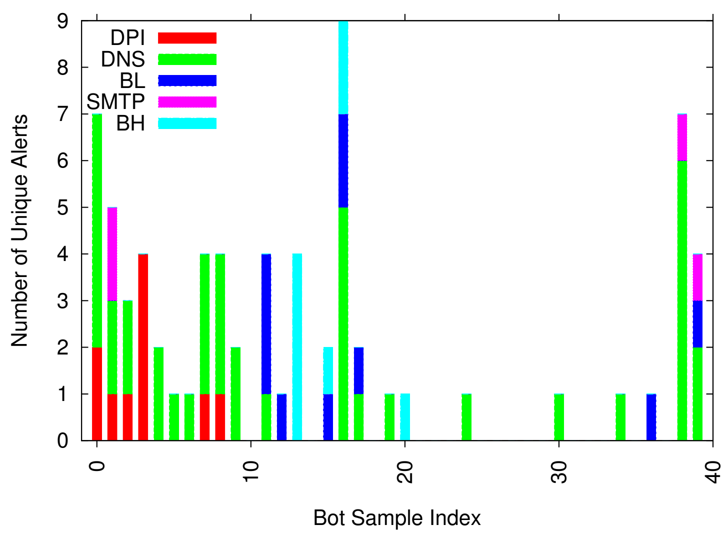

FIGURE 5.16. Suricata Unique Alerts per Bot with Encryption (MCFPEncryptedBots)

TABLE 5.7. Suricata Alerts of MCFPBotTraces and MCFPEncryptedBots

	Total Alerts	Unique Alerts
MCFPBotTraces	570155	291
MCFPEncryptedBots	306591	75
Encryption Damage	263564	216
Damage Percentage	46.2%	74.2%

Similar as the experiments with Snort in Section 5.4, now we use the number of detected bots as another criterion to evaluate the damage result from encrypted botnet traffic on Suricata. First, we measure the damage using the 140 bot traces. When we feed 140BotTraces and 140EncryptedBots to Suricata, 140 and 95 (32.1% less) bots are detected. Then we use the MCFP bot traces to measure the damage. 27 and 12 (55.6% less) bots are detected when MCFPBotTraces and MCFPEncryptedBots are used as inputs to Suricata, respectively.

In conclusion, encryption largely foils Suricata. For our two particular datasets, encrypted botnet communications 1) suppress more than 35% of total alerts and around 75% of unique alerts for Suricata, 2) make Suricata miss 32% and 55% of bots.

5.6. Summary

In this chapter, we first use BotTalker to generate sets of encrypted botnet traces and reverse engineer the event scores of BotHunter. After that, we evaluate the damage result from encrypted botnet traffic on three security systems, including BotHunter, Snort and Suricate. We have two findings from the experiments. One is that DPI technique plays a more important role than blacklist in BH. The other finding is that encryption largely foils these systems. For example, for dataset 140BotTraces, encrypted botnet communications make BH miss 125 out of 140 bots; suppress more than 70% and 75% of unique alerts for Snort and Suricata, respectively.

CHAPTER 6

Detecting Encrypted Botnet Traffic

In the previous chapter, we showed that encrypted payload can evade detection systems such as BH, Snort, and Suricata. The reason is that encryption makes the DPI rules blind. However, we notice that encryption does not foil the other detectors of BH that do not rely on DPI. A simple question then, is can we develop a detector to label encrypted traffic, and then combine it with the other detectors of BH to detect encrypted botnets? In this chapter, we attempt to answer this question by introducing two classifiers to detect high entropy (HE) flows and integrating them into BH to detect encrypted botnet traffic. The basis of this introduced method is that encryption tends to alter the payload entropy, often converting low entropy (LE) payload to HE.

6.1. Detecting High Entropy Data

In information theory, entropy was first introduced by C.E.Shannon [77] and it has been used widely in various domains. Here we present an introduction to entropy, followed by a brief description on how to select an HE threshold using the approach described in [39]. We use the latter to classify HE flows.

For a probability distribution $p = (p_i)_{i \in \Sigma}$, entropy is defined as

$$H(p) = -\sum_{i=1}^{n} p_i log p_i \tag{1}$$

Entropy can be applied to measure the uniformity of data. The more uniform the data, the lower its entropy is. Uniform distribution leads to the maximum entropy value. In our case, since we calculate the entropy of data composed of characters (bytes), n is $2^8 = 256$ in equation 1 and the maximum entropy value is 8 (bits per byte). If the distribution is highly skewed, the entropy is low. Consequently, we can use entropy to detect certain types of content. For example, high entropy may indicate encrypted or compressed data as the distribution is close to random. However, we face two challenges. The first is that $H(p)$ in the above equation is calculated based on an infinite amount or a closed set of input data. But in our case, the length of packet payload is less than 1500 bytes, which means p is *undersampled*. The second challenge is that given a chunk of data, we need to determine a threshold to classify HE or LE. We use the method introduced by Olivain [39] to solve the above two challenges. Specifically, we calculate entropy thresholds for N-bytes of data. Given a chunk of N-bytes of data, we calculate its entropy and compare it with the corresponding threshold. If the entropy is greater than the threshold, then it is HE data, otherwise it is LE. As it is not possible to calculate the threshold for infinite length of data, we calculate thresholds up to *N=640K*. For the data larger than 640KB, we use the threshold of *N=640K*. We believe this is reasonable because (a) the threshold for *N=640K* is very close to the maximum entropy value, and (b) the threshold increases very slowly when N is large.

6.2. High Entropy Flow Classifiers

Earlier we described a method to classify a chunk of data as HE or LE. Now we extend the methodology to classify a flow as HE or LE. We use the classic definition of a flow, namely a set of packets with the same source IP address and port, destination IP address and port and protocol.

Our first challenge is defining what a HE flow is. While many protocols encrypt user data, they exchange packets at the beginning of a flow that are not encrypted (and thus low entropy). After the initial exchange, subsequent packets are encrypted. Figure 6.1 illustrates this scenario.

FIGURE 6.1. Protocols Composed of Both Non-Encrypted and Encrypted Communications

The beginning part (gray) contains non-encrypted low entropy packets. To prevent such LE packets from skewing entropy calculation our algorithms wait until *N Sequential High Entropy Packets* have been detected before calculating entropy. We present two classifiers, a flow-based and a packet-based classifier. Both classifiers aim at labeling a flow as HE or LE, with the former examines the entire flow, while the latter examines each packet separately.

(1) Flow-based HE Classifier: The flow-based classifier begins by calculating the entropy of each packet. After detecting N sequential HE packets the classifier examines the payload of all subsequent packets and then calculates the cumulative entropy of all the data (including the initial HE packets). The classifier then compares the cumulative entropy with the appropriate threshold, as described earlier to make the final decision.
(2) Packet-based HE Classifier: After detecting N sequential HE packets this classifier calculates the entropy of each packet and classifies it as HE or LE. At the end of the flow we count the number of HE and LE packets, denoted as *N(HE)* and *N(LE)*. If *N(HE)/(N(HE)+N(LE))* is greater than the *High Entropy Packet Percentage Threshold*, then the flow is classified as HE.

If classifiers do not detect N sequential HE packets the flow is classified as LE. We determined N experimentally and found that $N = 3$ works best.

6.3. Comparing Flow and Packet-based HE Classifiers

Now we compare the flow and packet-based HE classifiers with three tests. In the first test, we run both HE classifiers on encrypted files to validate our assumption that encrypted data is typically HE. In the second test, we run our classifiers on encrypted network flows, specifically HTTPS flows, and expect most of them to be classified as HE. Finally, in the third test we run the classifiers on known non-encrypted (but not necessarily LE) traffic to ensure we do not see any anomalous results.

(1) Test 1: Encrypted Files

We use 17,000 encrypted files in data set EncryptedFiles in this experiment. We use N = 3 as sequential HE packets and 90% as the HE packet percentage threshold. The results show that both flow and packet-based classifiers identify all the encrypted files as HE. This is an encouraging result, indicating that encrypting a file does result in HE.

(2) Test 2: Encrypted Traffic

In this test we repeat the above methodology, but this time using real encrypted network traffic, specifically the trace *Lab1-HTTPS*. The results are shown in Table 6.1. As we can see from the table the two classifiers identify 96% and 99% of the flows in the trace as HE, a very good result.

(3) Test 3: Non-encrypted Files

We carry out the experiment using the data set NonEncyrptedFiles, which includes 11 types of non-encrypted files, and 1000 files per type. Table 6.2 shows the

Table 6.1. Online Encrypted Traffic

Traffic	HTTPS
Total Flows	1675
HE (flow-based)	1612
Rate (flow-based)	96.24%
HE (packet-based)	1665
Rate (packet-based)	99.40%

results. We see that none of the Txt files is classified as HE by either classifier. For the compressed files, as expected, many are classified as HE by both classifiers.

Our conclusion is that the packet-based classifier seems more eager to consider data as HE than the flow-based classifier. One explanation is that many files (e.g., compressed files) may have structures and contain duplicate or similar contents in each "block". Consequently, considering the entire file, there is increased duplication and a decrease in entropy. On the contrary, a single packet may not contain such duplicate.

Table 6.2. Offline Non-Encrypted Files

File Type	HE Files (flow-based)	HE Files (packet-based)	File Type	HE Files (flow-based)	HE Files (packet-based)
DOC	0	31	MP3	0	0
EXCEL	0	20	bz	107	515
TXT	0	0	gz	14	709
PDF	0	1	Rar	70	752
JPG	0	232	zip	17	705
Executable	422	534			

6.4. Enhancing BotHunter with a High Entropy Detector

Armed with two HE classifiers, we move toward enhancing BotHunter with an additional event that enables the detection of the HE flows. The hope is that the presence of HE flows in conjunction with existing bot events that BH already detects, will flag bots using encrypted traffic. In this section we first outline our modifications to BH, and then use encrypted botnet traces to determine true positives and our lab traces to determine false positives.

We enhance BH as follows. First, we implement a HE flow detector separate from BH since we do not have access to BH's source code to integrate our code. For the HE flow detector, we choose the packet-based classifier. While this classifier tends to flag more flows as HE than its flow-based counterpart, we feel that BH will suppress many of the false positives. Moreover, the flow-based HE classifier requires more memory than the packet-based one for each flow to accumulate packet payloads. Our detector adds an entry to the Snort log when it detects at least one HE flow between two hosts. We then resort to a hack, where our detector triggers an existing event in BH by adding the appropriate entry in the Snort log, namely the C&C communications or egg download event. Recall from Section 5.2 that both these events have a score of 0.5, which we deem as the upper bound for our HE detector. The reason is that the C&C communication and egg download event are triggered using DPI, which carries a much higher confidence than our detector. In our experiments we try score values from 0.1 to 0.4.

We carry out two experiments to select the proper score for the entropy detector. We use the encrypted botnet data set 140EncryptedBots to evaluate true positives. The network traces Lab1, Lab2 and Lab3 are used to evaluate false positives. The results are shown in

Figure 6.3 and Figure 6.2. The x-axis in both figures is the score assigned to the HE detector and the y-axes are a count of true and false positives respectively.

In Figure 6.2, when the score of the HE detector is 0.2 we notice one potential false positive in Lab1. The reason is that events C&C communication based on DPI and C&C communication based on blacklist are triggered, whose total score is 0.6. At the same time, this host also has HE connections, so the HE detector is triggered and the final score is greater than threshold. As a result, this host is flagged as a bot. Figure 6.3 shows that the enhanced BotHunter is able to detect 134 out of the 140 known bots when the weight of the detector is 0.2. We thus conclude that a weight value of 0.2 works best.

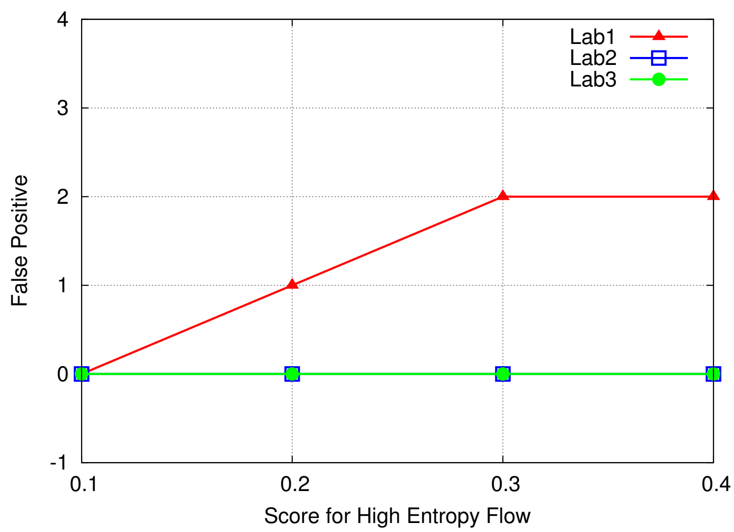

FIGURE 6.2. False Positives for Lab Traces

6.5. Discussion

Botnet can use any encryption scenarios they choose to evade detection system. As rolling key encryption is used by many legitimate applications, it is worth to discuss whether our method will be foiled if the botnets use rolling key encryption. Our method detects encrypted botnets by looking for HE traffic and also other malicious events that can be

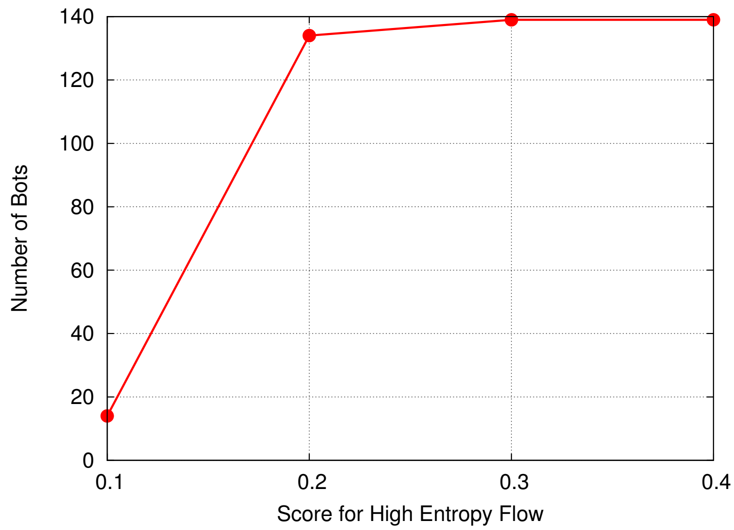

FIGURE 6.3. True Positives for bot traces

detected by BotHunter. Even the botnets use rolling key encryption, the encrypted traffic will still be labeled as HE flow by our HE classifiers. Then, if enough other suspicious activities are detected by BotHunter, the bots can still be detected.

In Section 6.4, we use 140EncryptedBots to decide the score of HE classifier. An important question to ask is that whether the score decided by these 140 botnet traces applies to many other botnets? Before answering this question, we want to emphasize that instead of setting specific score, the main contribution of our work in this section is to show that entropy helps detect encrypted botnet traffic and we have built two HE classifiers to do this. We admit that the score decided by these 140 traces may not apply to some other botnets. But we also notice that there is no single system can detect all the botnets. We plan to obtain more encrypted botnet traces to decide the score. Another way to validate the score with small number of botnet traces is to use k-fold cross-validation.

In the experiments, we use separate legitimate lab traces and botnet traces to evaluate the false positive and true positive. However, in the real world network traffic, the botnet and legitimate traffic is mixed, which may make botnet detection much harder. One question we need to answer is that what the false positive and true positive of our method will be if we use such mixed traffic as input, for example, if we blend 140EncryptedBots with Lab1, Lab2 and Lab3. Our answer is that we will get the same results as we run the method on the legitimate the botnet traces separately. The reason is that the enhanced BotHunter detects bots by analyzing the traffic from a single host, instead of the traffic collected from multiple hosts.

6.6. Summary

In this chapter we investigate detection of bots that use encrypted communication. We first show that encryption in botnet communication increases entropy. Then, we introduce flow-based and packet-based classifiers to detect HE flows. Both classifiers are able to skip the unencrypted part and detect the following encrypted communications. Finally, we add packet-based HE classifier to BotHunter and make it be able to detect encrypted bots.

CHAPTER 7

Early Detection of High Entropy Traffic

In the previous chapter we introduced two classifiers to label a flow as HE or LE. However, entropy calculation is expensive, especially for long flows. A simple question then, is whether it is actually required to examine the entire flow, or can we reach the same classification if we look at part of the flow and how much. In this chapter, we first improve our previous classifiers by turning them into *early high entropy classifiers*. In particular, we show we can classify a flow as HE or LE by looking at the first M packets, rather than the entire flow. This saves both CPU and memory resources, enables faster detection and higher throughput. Then, we demonstrate how our early HE classifiers can improve the performance of IDS when used as *filters*.

7.1. Early HE Classifiers

The early HE classifiers are similar as the HE classifiers proposed in Section 6.2. Instead of looking at the entire flow, we look at the first M packets to label a flow as HE or LE, as shown in Figure 7.1. The early flow-based and packet-based HE classifiers are as described as follows.

(1) Flow-based Early HE Classifier: After detecting N sequential HE packets we capture the payload of all subsequent packets until the M^{th} packet, and calculate the cumulative entropy of the resulting data. We then compare the cumulative entropy with the threshold and if the cumulative entropy is greater than the threshold, then the flow is labeled as HE.

(2) Packet-based Early HE Classifier: After detecting N sequential HE packets, we calculate the entropy for each packet until the M^{th} packet, and classify each of them as HE or LE. We count the number of HE and LE packets, denoted as *N(HE)* and *N(LE)*. If *N(HE)/(N(HE)+N(LE))* is greater than HE packet percentage threshold, then we consider the flow as HE.

In both classifiers, if N sequential HE packets are not detected in the first M packets, the flow is labeled as LE.

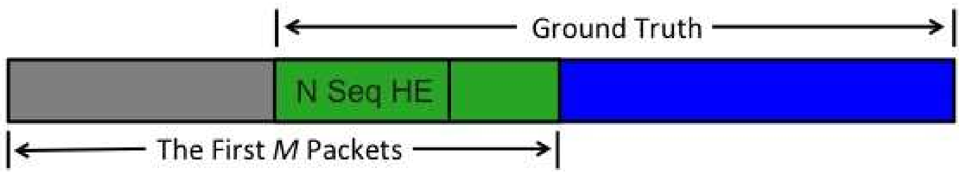

FIGURE 7.1. Early Classification of High Entropy Flows

7.2. EVALUATION

To carry out our evaluation we first classify our datasets into HE and LE using the full flow-based classifier described earlier in Section 6.2. We consider this as our ground truth. Then we run the early HE classifiers on HE flows and LE flows, and calculate the *recall* of HE and LE flows. Recall of HE flows is calculated as the percentage of HE flows classified correctly when using the early classifiers. Recall of LE flows is defined similarly. We evaluate the early HE classifiers using network traffic first followed by application content. Finally we drill down to encrypted vs. non-encrypted content separately.

7.2.1. EVALUATION WITH NETWORK TRAFFIC. First we evaluate the early HE detection classifiers using the Lab1-HTTPS dataset. Using the full flow-based classifier 1612 out of 1675 (96.24%) flows are labeled as HE.

For the packet-based early HE classifier, we use 9 different values for the HE packet percentage threshold, ranging from 10% to 90%. The recall of HE flows for flow-based and packet-based early HE classifiers is shown in Figure 7.2. The x-axis is the number of packets used to make decision and the y-axis is the recall. From the results we can see that both flow-based and packet-based early HE classifiers perform well in detecting HE flows. Specifically, the recall increases very quickly, over 90% at the seventh packet and almost 100% after the tenth packet. The reason why we need 10 packets to achieve 100% recall of HE flows is that the first several packets in the flows are protocol related negotiation in the clear, preventing us from detecting 3 sequential HE packets. We also find that there is no significant difference in recall by varying the HE packet percentage threshold. The reason is that most of HTTPS packets are encrypted and thus classified as HE, so even with a high threshold (e.g., 90%) the flow is still classified as HE.

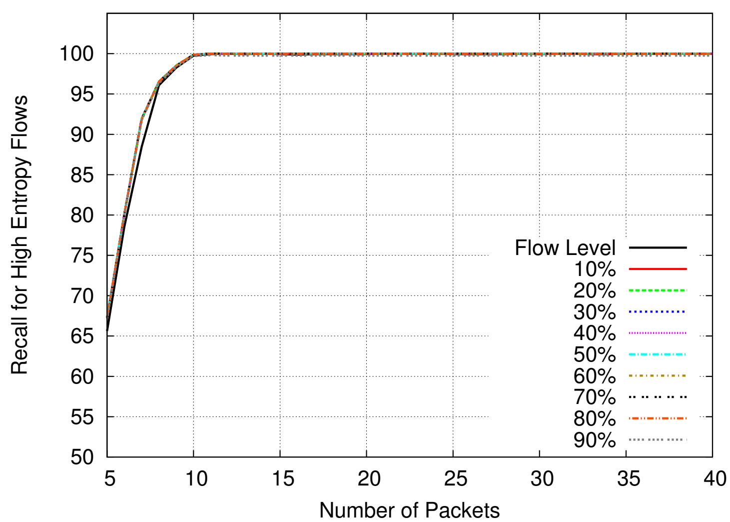

FIGURE 7.2. Recall of HE HTTPS Flows using Flow and Packet-based Early HE Classifiers

7.2.2. EVALUATION WITH APPLICATION CONTENT. In this section we use application content that includes 11,000 non-encrypted files and 17,000 encrypted files, to evaluate the early HE classifiers. Then in Section 7.2.3 and Section 7.2.4, we evaluate the classifiers on encrypted and non-encrypted content respectively. After applying the full-flow ground truth classifier, 10370 files are labeled as LE and the other 17630 files are labeled as HE.

First we evaluate the packet-based early HE classifier with 9 different HE packet percentage thresholds on the set of HE and LE files. The results of recall of HE files are shown in Figure 7.3. The figure shows that i) the recall of HE files is 97% even we only consider the first 5 packets, and ii) the HE packet percentage threshold has little effect. Figure 7.4 depicts the recall of LE files. We can see that as the HE packet percentage threshold increases, the recall also increases. The reason is that the packet-based classifier requires more HE packets to classify a file as HE, which means requiring fewer LE packets to classify a file as LE. In summary, with the packet-based early HE classifier using 90% as HE packet percentage threshold and 10 as M, the recall of HE files and LE files is 97% and 65% respectively.

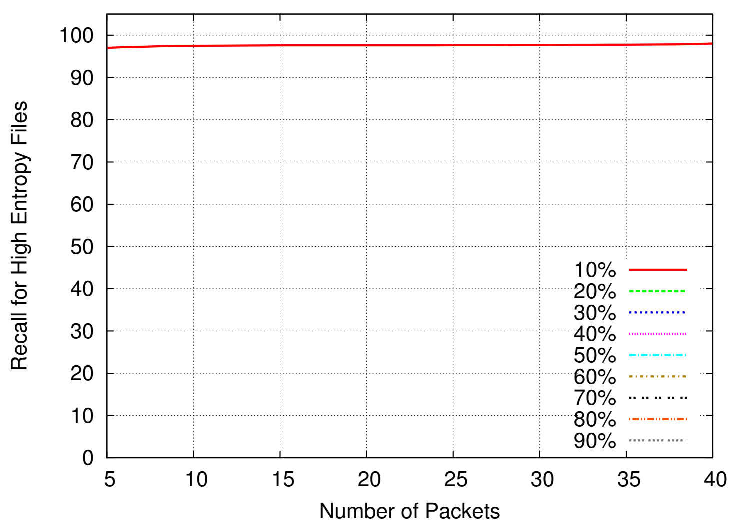

FIGURE 7.3. Recall of All HE Files using Packet-based Early HE Classifier

Next we apply the flow-based early HE classifier on the ground truth files. Figure 7.5 depicts the results. There are two observations we make. The first is the recall of HE files is very high, above 97% if we consider more than 5 packets. The second is that as the number

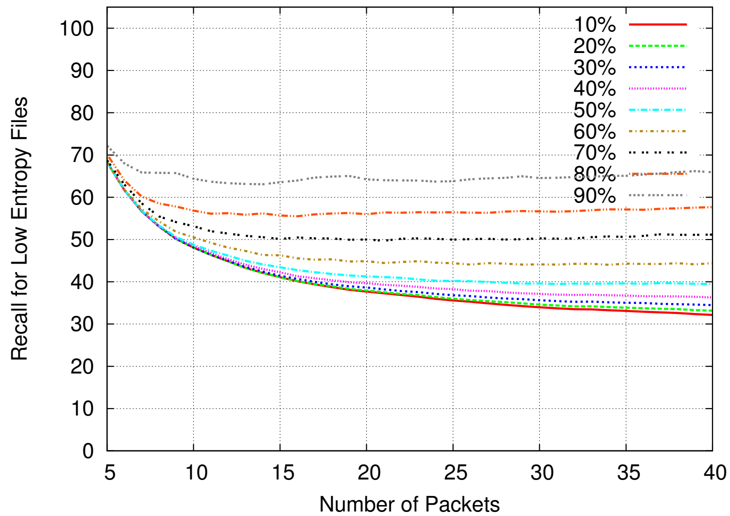

FIGURE 7.4. Recall of All LE Files using Packet-based Early HE Classifier

of considered packets increases, the recall for LE files increases. If we use 10 packets as M, the recall of HE files and LE files is 97% and 75%, respectively. Consequently, the flow-based early HE classifier performs better than its packet-based counterpart.

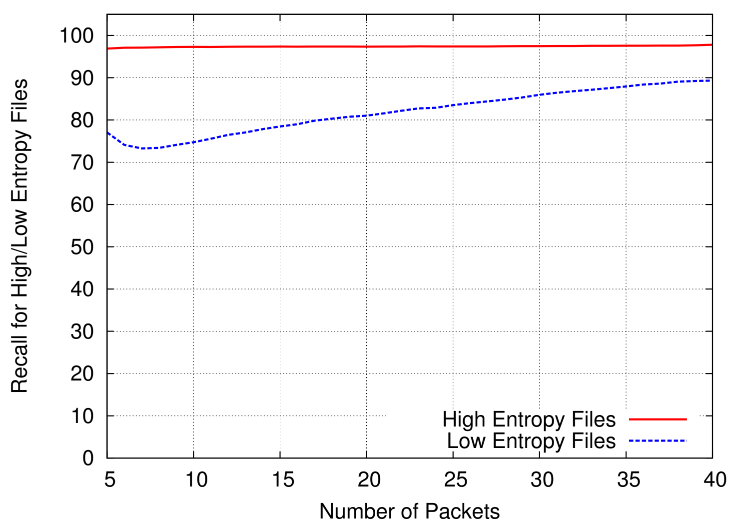

FIGURE 7.5. Recall of All HE and LE Files using Flow-based Early HE Classifier

7.2.3. EVALUATION WITH ENCRYPTED APPLICATION CONTENT. In this section, we use dataset EncryptedFiles to evaluate the early HE classifiers. All the 17,000 files are labeled as HE by our full-flow classifier. The recall of HE encrypted files using flow-based and packet-based early HE classifiers with 9 different HE packet percentage threshold values is over 99.9% of the time, even with only 5 initial packets. The reason for the high success rate is that all 17 encryption algorithms we used are strong. In summary, the early HE classifiers perform very well on encrypted files.

7.2.4. Evaluation with Non-Encrypted Application Content. Now we use dataset NonEncryptedFiles to evaluate the early HE classifiers. Figure 7.6 depicts the recall of HE and LE content by using the flow-based early HE classifier. The figure shows that as we consider more packets, the recall of LE and HE files increase. However, the recall for HE content is only around 30%. Upon further investigation, we find the reason is that for many executable files the 3 sequential HE packets start late, sometimes after dozens or hundreds of packets.

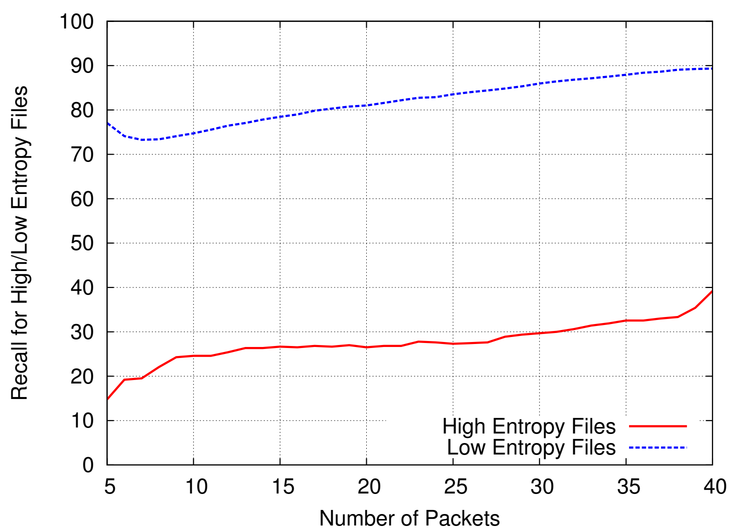

Figure 7.6. Recall of HE Non-Encrypted Files (include executable files) using Flow-based Early HE Classifier

We quantify where the 3 sequential HE packets start in a CDF in Figure 7.7. The x-axis is the packet offset from the beginning of the file where the 3 sequential HE packets start and the y-axis is the cumulative percentage. Note that at $x = 0$, y is the percentage of files that do not contain 3 sequential HE packets. For example, none of the txt files (green line) contains 3 sequential HE packets. The bottom orange line stands for exe files and shows that around 55% of the exe files do not contain 3 sequential HE packets, or they start before the 40th packet. However, for the other 45% of exe files the 3 sequential HE packets start after the 40th packet, and for 22% of exe files after the 100th packet. Consequently, we miss these late 3 sequential HE packets in exe files when we only consider the first 40 packets or less, and the files are classified as LE. Upon further investigation we discover that these exe files contain packed or compressed code, so the 3 sequential HE packets start late. We use software *Exeinfo PE* [7] to view the internal structure of the exe files and find that at least 520 exe files contain self-extracting archive (SFX) or a compressed archive. If we exclude the exe files and plot the recall for the other 10 file types as shown in Figure 7.8, the recall of HE files is much higher, around 80% for both HE and LE files if we consider the first 15 packets.

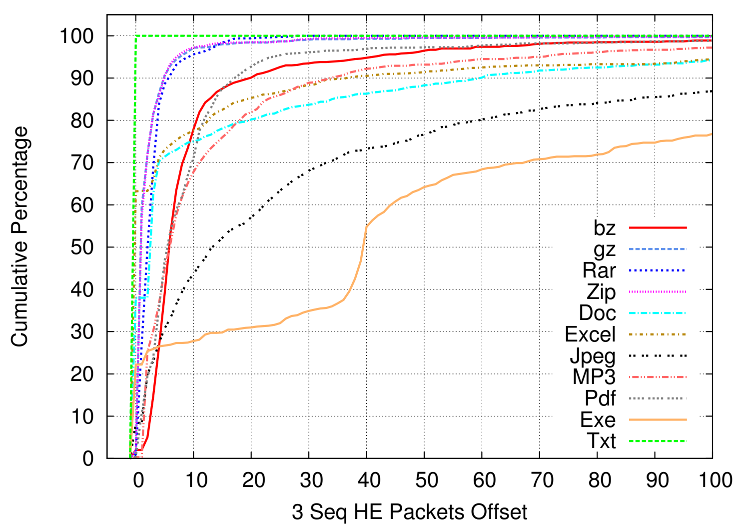

FIGURE 7.7. CDF of the Offsets of 3 Sequential HE Packets

The above results show that for the data whose N sequential HE packets start late, our early HE classifiers may be suffered. To deal with this issue, instead of the first M packets

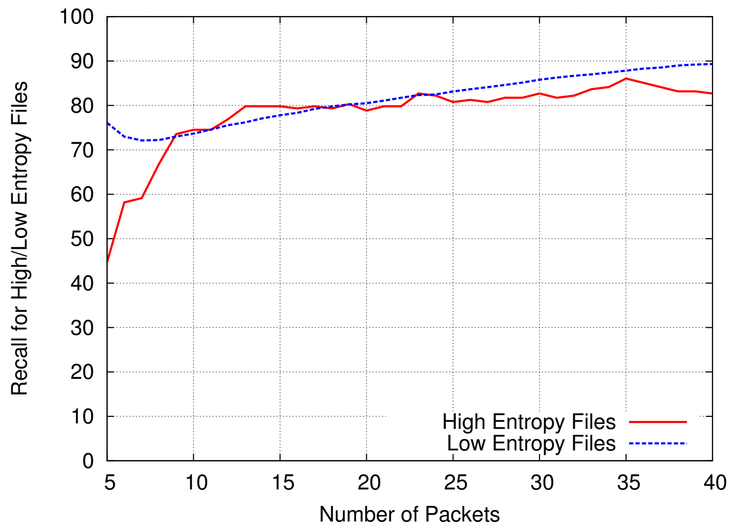

FIGURE 7.8. Recall of HE Non-Encrypted Files (exclude executable files) using Flow-based Early HE Classifier

from the beginning of a flow, we can use the first M packets after the N sequential HE packets to classify a flow. In this case, however, we need to examine more packets.

Finally, we evaluate the recall for the packet-based early HE classifier on the non-encrypted files. Figure 7.9 shows the recall for HE files. We see similar results with exe files as the flow-based classifier, the recall for HE content is only around 30%.

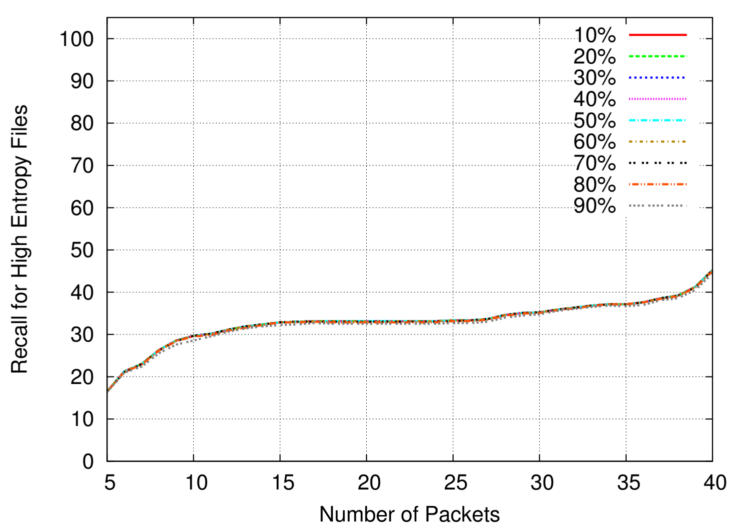

FIGURE 7.9. Recall of HE Non-Encrypted Files (include executable files) using Packet-based Early HE Classifier

The recall for the other 10 types of files after excluding exe files is shown in Figure 7.10. The recall is close to 100% after the 15th packet and insensitive to the HE packet percentage threshold.

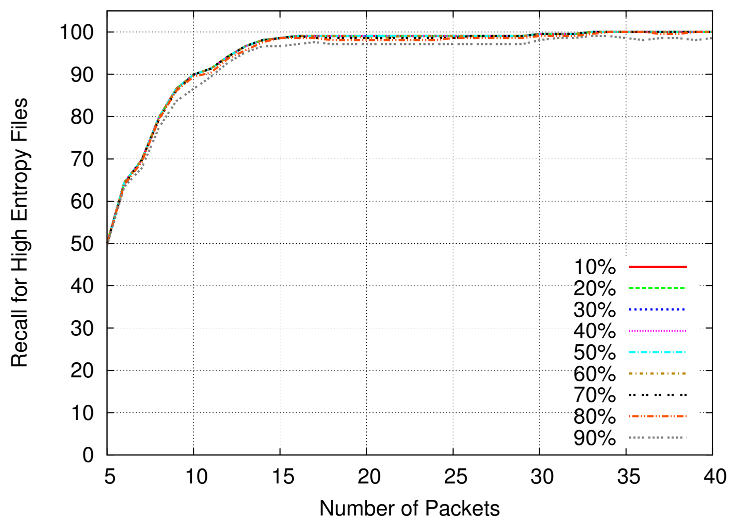

FIGURE 7.10. Recall of HE Non-Encrypted Files (exclude executable files) using Packet-based Early HE Classifier

The recall with LE files including exe is depicted in Figure 7.11. We can see that as the HE packet percentage threshold increases, the recall also increases. The recall of LE files after excluding exe files is almost the same as with exe files, so we omit the figure. LE exe files pose no problem, they are always classified as LE in flow-based or packet-based early classifiers when we consider the first few packets.

7.3. Reducing Load on IDS

Previously, we introduced two early HE classifiers for labeling HE and LE data. Now, we apply them to reduce the amount of traffic delivered to an IDS. Many IDSs rely on DPI techniques, such as searching for suspicious strings in packet payload. However, certain content, such as video, audio, encrypted or compressed data, cannot be understood by the

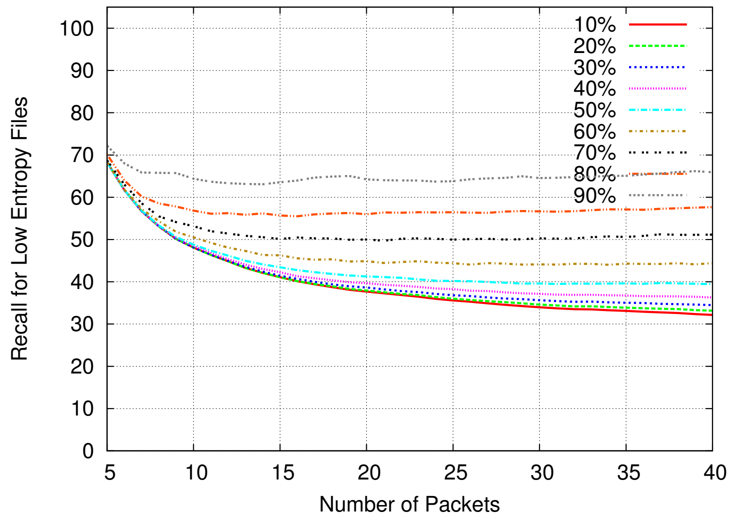

FIGURE 7.11. Recall of LE Non-Encrypted Files (include executable files) using Packet-based Early HE Classifier

IDS and results in a waste of computation resources. To address this issue, we investigate the use of our early HE classifiers to label traffic as *clear* or *opaque*. Then, we only deliver the clear traffic and the first several packets of opaque traffic to the IDS and discard the rest.

7.3.1. EARLY CLASSIFICATION OF CLEAR DATA AND OPAQUE DATA. We evaluate the ability of early HE classifiers to distinguish clear from opaque data. Specifically, we designate flows labeled as HE by the early classifier as opaque and LE flows as clear. Note that this is different from classifying data as HE or LE. **We are evaluating the ability of early HE and LE classifications to imply opaque and clear data, respectively.** First, we analyze the NonEncryptedFiles dataset. We consider txt files to be clear, and bzip, gzip, rar, winzip, jpeg, mp3, pdf and exe files to be opaque. We do not include doc and excel files as either clear or opaque because they could consist entirely of text or images. The former is clear while the latter is opaque. As a result, it is hard to classify doc and excel files without manual analysis.

From an implementation point of view, the flow-based early HE classifier requires more memory than the packet-based one for each flow to accumulate packet payloads. Consequently, we use the packet-based early HE classifier and a 90% HE packet percentage threshold for the traffic reduction method. The percentages of files classified as opaque are shown in Figure 7.12. All the txt files are classified as clear. If we limit classification to the first 15 packets, 65.5% to 81.4% of compressed files are labeled as opaque, and 6.5% to 27% of jpeg, mp3, pdf and exe files are labeled as opaque. Surprisingly, many files that are labeled as LE by the full classifiers are instead labeled as HE when the early classifiers are applied. However, because we define clear and opaque data based on our early classifiers, thus these files are considered to be opaque. One explanation is that these files have structures and contain duplicate or similar contents in each "block". Consequently, considering more packets, there is increased duplication and a decrease in entropy. Next, we repeat the experiment on the EncryptedFiles dataset, and all files are classified as opaque. In summary, 20,667 out of 25,000 (82.67%) opaque files are correctly labeled by the packet-based early HE classifier.

7.3.2. Network Traffic Reduction. We have shown that the early HE classifiers can be used to distinguish clear and opaque data. In this section we describe how to use them to reduce the amount of traffic delivered to IDS.

Our *traffic reduction* methodology is as follows. For a given flow, we use the first M packets to classify the flow as HE or LE. If it is classified as HE, then we consider this flow to be opaque and do not deliver any of the following packets to the IDS. If the flow is classified as LE, then we consider this flow to be clear and deliver all packets to the IDS. Note that the first M packets of both clear and opaque flows are delivered to the IDS. In this

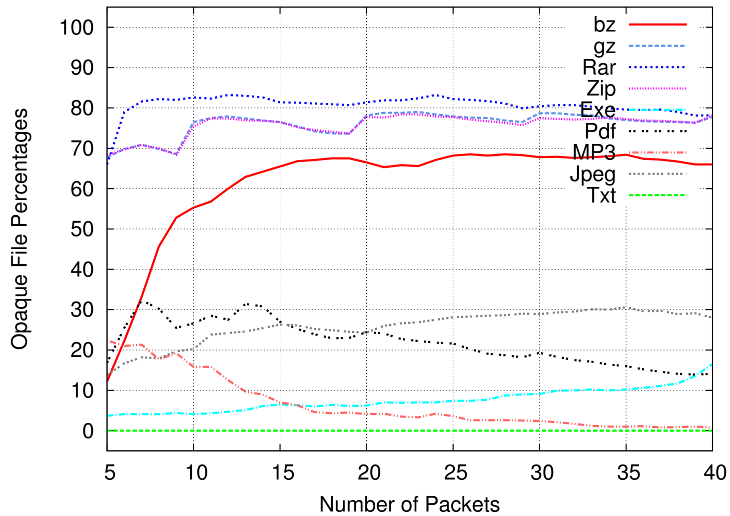

FIGURE 7.12. Opaque File Percentages for Non-Encrypted Files using Packet-based Early HE Classifier

implementation we use the packet-based early HE classifier, we set the HE packet percentage threshold to 90%, and M to be 15.

We now investigate how much traffic is reduced by using our traffic reduction method. In particular, we look at the *byte reduction rate* and *packet reduction rate* to quantify the amount of reduced traffic. The byte reduction rate is calculated as the number of bytes discarded after the M^{th} packet in opaque flows divided by the total bytes for all the opaque and clear flows. The packet reduction rate is defined similarly. We use TCP and UDP traffic in trace Lab1, Lab2, and Lab3, and all the packets in CSUTrace for measurement. The results are shown in Table 7.1 and Table 7.2. From the tables we can see that our methodology prevents a great deal of packets and bytes from being delivered to IDS. For example, in Lab2 and Lab3, the traffic is reduced for more than 50%.

While our methodology greatly reduces traffic to the IDS, the central question is, have we also thrown away traffic that would trigger alerts?

Table 7.1. Byte Reduction Rate

Dataset	Total Bytes	Reduced Bytes	Reduced Rate
Lab1	134230334179	18428005888	13.73%
Lab2	143939409963	83873010412	58.27%
Lab3	403369729161	213819208748	53.01%
CSUTrace	8140578402030	2772424035389	34.06%

Table 7.2. Packet Reduction Rate

Dataset	Total Packets	Reduced Packets	Reduced Rate
Lab1	102651430	13613902	13.26%
Lab2	122491839	62150940	50.74%
Lab3	303501833	156448899	51.55%
CSUTrace	8942932452	2999274638	33.54%

To answer this question, we use Snort and the Emerging Threats Open Ruleset in our evaluation. Snort provides a module named preprocessor, which can be used to extend functionality by allowing users to add modular plugins into Snort. We implement our traffic reduction algorithm as a preprocessor and integrate it into Snort.

To investigate how many alerts are missed by applying the traffic reduction preprocessor, we compare alerts from the original version of Snort and with the preprocessor on the same trace. As the trace CSUTrace does not contain packet payload, we only use the three lab traces in this experiment. Table 7.3 shows the results of the two versions of Snort. The table shows that the version of Snort with traffic reduction preprocessor is able to detect more than 99.9% of the alerts detected by the original Snort. Thus we conclude that our traffic reduction has virtually no effect on the IDS, we have essentially thrown away traffic would not have triggered alerts.

In a second experiment we use BotHunter to evaluate the performance of the traffic reduction algorithm on its ability to detect bots. We first feed the botnet traces 140BotTraces

Table 7.3. Triggered Alerts with/without Traffic Reduction (TR) Preprocessor

Data set	Alerts without TR	Alerts with TR	Missed Alerts	Detection Rate
Lab1	8184	8180	4	99.95%
Lab2	42570	42552	18	99.96%
Lab3	63004	63003	1	99.99%

to BH and get a detection rate of 138/140 bots. Then we integrate the traffic reduction preprocessor into BH and feed the bot traffic again. The results remain unchanged at 138/140, meaning that our traffic reduction has no effect on BH.

7.3.3. Comparison with Previous Work. **Comparison with Classification of Clear and Opaque Packets:** The work most related to ours is White et al. [94], where the authors use *Sequential Probability Ratio Test* (SPRT) to label a packet as opaque or clear. Then, only clear packet is delivered to IDS and the opaque is discarded. The main difference with our work is that they focus on individual packets, thus they have to examine every packet. On the contrary, our aim is to classify opaque flows. We only use up to the first 15 packets to classify a flow as opaque or clear, and then only clear flows are delivered to IDS. We now investigate how many packets are used to classify flows as clear or opaque. We use TCP and UDP traffic in trace Lab1, Lab2, and Lab3, and all the traffic in CSUTrace in the experiment. The results are shown in Table 7.4. From the table we can see that only less than 8% of total packets are used to classify all the flows, meaning that the method in [94] inspects at least 13 times more packets than ours.

Comparison with Byte-based Per-flow Cutoff Algorithms: Some past work filters traffic to an IDS by only delivering the first n bytes of each flow to IDS [62], [55]. For example, Maier et al. [62] use the first 10−20 KB as per-flow cutoff. We call these methods *byte-based*

Table 7.4. Packets for Classifying Clear/Opaque Flows

Data set	Total Packets	Packets to Classify	Percent
Lab1	102651430	4808830	4.68%
Lab2	122491839	8316488	6.79%
Lab3	303501833	7211991	2.38%
CSUTrace	8942932452	695423897	7.78%

per-flow cutoff algorithms. Their assumption is that the early stage of each flow triggers most alerts. However, the later part of a flow may still contain important information. First we investigate how many alerts are triggered by byte-based per-flow cutoff algorithms. We use 20KB as per-flow cutoff. The results are shown in Table 7.5. From the table we can see that around 95% alerts are triggered, while our traffic reduction algorithm triggers more than 99.9% alerts (Section 7.3.2). To compare byte-based per-flow cutoff algorithms with ours more straightforward, we list the number of missed alerts in Table 7.6. From the table we can see that the byte-based per-flow cutoff algorithms miss at least 71 times of alerts more than ours. In the worst case, they miss 3112 alerts while we only miss 1.

Table 7.5. Triggered Alerts of Byte-based Per-flow Cutoff Algorithms

Data set	Alerts (without cutoff)	Alerts (with cutoff)	Rate
Lab1	8184	7695	94.02%
Lab2	42570	41278	96.96%
Lab3	63004	59892	95.06%

Table 7.6. Comparison of Missed Alerts of Byte-based Per-flow Cutoff Algorithms and Traffic Reduction (TR)

Data set	Missed Alerts (TR)	Missed Alerts (cutoff)	Rate
Lab1	4	489	122.25
Lab2	18	1292	71.78
Lab3	1	3112	3112

Comparison with Packet-based Per-flow Cutoff Algorithms: Similar as byte-based per-flow cutoff algorithms, some work only delivers the first m packets of each flow to IDS [28], [67]. We call them *packet-based per-flow cutoff* algorithms. Similar as above, we first investigate the number of alerts triggered by packet-based per-flow cutoff algorithms with 20 packets as the limit. The results are listed in Table 7.7, which tells that around 95% alerts are triggered. Then we compare the number of missed alerts of packets-based per-flow cutoff algorithms with ours. The results are shown in Table 7.8. From the table we can see that the packet-based per-flow cutoff algorithms miss at least 79 times of alerts more than ours. In the worst case, they miss 3326 alerts while we only miss 1.

TABLE 7.7. Triggered Alerts of Packet-based Per-flow Cutoff Algorithms

Data set	Alerts (without cutoff)	Alerts (with cutoff)	Rate
Lab1	8184	7610	92.99%
Lab2	42570	41132	96.62%
Lab3	63004	59678	94.72%

TABLE 7.8. Comparison of Missed Alerts of Packet-based Per-flow Cutoff Algorithms and Traffic Reduction (TR)

Data set	Missed Alerts (TR)	Missed Alerts (cutoff)	Rate
Lab1	4	574	143.5
Lab2	18	1438	79.89
Lab3	1	3326	3326

7.4. DISCUSSION

An attacker can bypass the IDS by prepending opaque traffic to the communication stream. This, however, is a general limitation of all techniques, not just ours, that attempt to arrive at early conclusions without looking at all the data. One way to counteract such evasion is to pick packets within an HE-classified flow at random and test for LE. Flows flagged this way can be placed on an observation list. Similarly, an attacker can bypass the IDS by encrypting or compressing all the communications. Note that in the case of bot detection using a tool such as enhanced version of BotHunter these evasive actions will fail - an HE stream will trigger a BotHunter alert.

Our methods only improve efficiency from the perspective of software. Besides that, hardware might also be used to accelerate entropy calculation and DPI rules match. However, even if the IDS can keep up with traffic speed with hardware help, delivering data that cannot be understood by IDS is still a waste of resources because more than 99.9% alerts can be detected by our traffic reduction algorithm. Moreover, by using only the first 15 packets we can label HE flows and detect bots at the early stage, where hardware does not help too much.

7.5. Summary

In this chapter we introduce two early HE classifiers to classify flows as HE or LE by looking at only a portion of a flow, which improves classification performance. Using the early HE classifiers, we show benefits in bot detection and develop a traffic reduction algorithm to reduce the load on an IDS and implement it as a Snort preprocessor. Our results show that we can reduce the traffic delivered to an IDS by more than 50% while maintaining more than 99.9% of the original alerts. We also compare our traffic reduction scheme with previous related work. We find that they need to examine at least 13 times more packets than ours or they miss at least 70 times of the alerts.

CHAPTER 8

BotDigger: Detecting DGA Bots in a Single Network

In this chapter, we introduce BotDigger, a system that detects an ***individual DGA-based bot*** by only using DNS traffic collected from a ***single network***. Note that "detecting individual bot in a network" does not mean BotDigger cannot detect all the bots in a network. If there are multiple bots in the same network, BotDigger can still detect them, but individually.

8.1. Methodology

8.1.1. Notations.

Domain names are organized in a tree-like hierarchical name space. A domain name is a string composed of a set of labels separated by the dot symbol. The rightmost label is called top-level domain (TLD). Two commonly used TLDs are generic TLD (gTLD, e.g., .com) and country code TLD (ccTLD, e.g., .cn). A domain can contain both gTLD and ccTLD, for example, *www.foo.example.com.cn*. In this paper, we consider the consecutive gTLD and ccTLD as a single TLD for simplicity. We define the domain that is directly to the left of the TLDs as *Second Level Domain* (2LD), and define the domain to the left of the 2LD as *Third Level Domain* (3LD). The 2LD and 3LD in the above example domain is "example" and "foo", respectively. Note that in this work we denote a 2LD of a NXDomain as *2LDNX*.

8.1.2. System Overview. An overview of the methodology is shown in Figure 8.1. First, several filters (Section 8.1.4) are applied to remove unsuspicious NXDomains (e.g., the domains with invalid TLDs). The remaining suspicious NXDomains are then grouped by host who sends the queries. Note that in the following steps, we focus on the queried domains from each individual host, and that is the reason why our method can detect individual bot. Now the *quantity evidence* (Section 8.1.5) is applied to extract outliers in terms of the number of queried suspicious 2LDNXs. For each outlier, we use the *temporal evidence* (Section 8.1.6) to extract the period of time when a bot begins to query DGA domains until it hits the registered C&C domain, denoted as (t_{begin}, t_{end}). If such period cannot be extracted, then the host is considered as legitimate, otherwise the host is considered suspicious and the following analysis is performed. The next step focuses on the suspicious NXDomains being queried between t_{begin} and t_{end}. The linguistic attributes of these NXDomains are extracted and then the *linguistic evidence* (Section 8.1.7) is applied on the extracted attributes. The assumption of linguistic evidence is that a bot queries many suspicious NXDomains that have similar linguistic attributes thus they will likely be clustered together. In particular, a hierarchical clustering algorithm is applied on the attributes. The output is one or more clusters of linguistic attributes and the corresponding suspicious NXDomains. We consider the clusters whose sizes are greater than a threshold (named *BotClusterThreshold* as defined in Section 8.1.7.3) as bot NXDomain cluster candidates. If all the clusters of a host are smaller than the threshold, then the host is considered legitimate. Finally, to identify the C&C domains, the DGA domain signatures are extracted from the bot NXDomain cluster candidates and matched against all the successfully resolved domains queried between t_{begin} and t_{end}. The domains that match the signatures are considered as C&C domain candidates, and the host is labeled as a bot. The host is not labeled if no C&C domain candidate can be extracted. Note that we do not precisely label C&C domain. Instead, we label multiple C&C domain candidates. It is possible, however, unlikely, that a successful request is done by the infected host right after a series of failed requests, and the legitimate domain in the request is close in lexicographic distance. For this reason we can not be absolutely certain that a successful DNS request is a C&C server. Precisely labeling single C&C domain is future work.

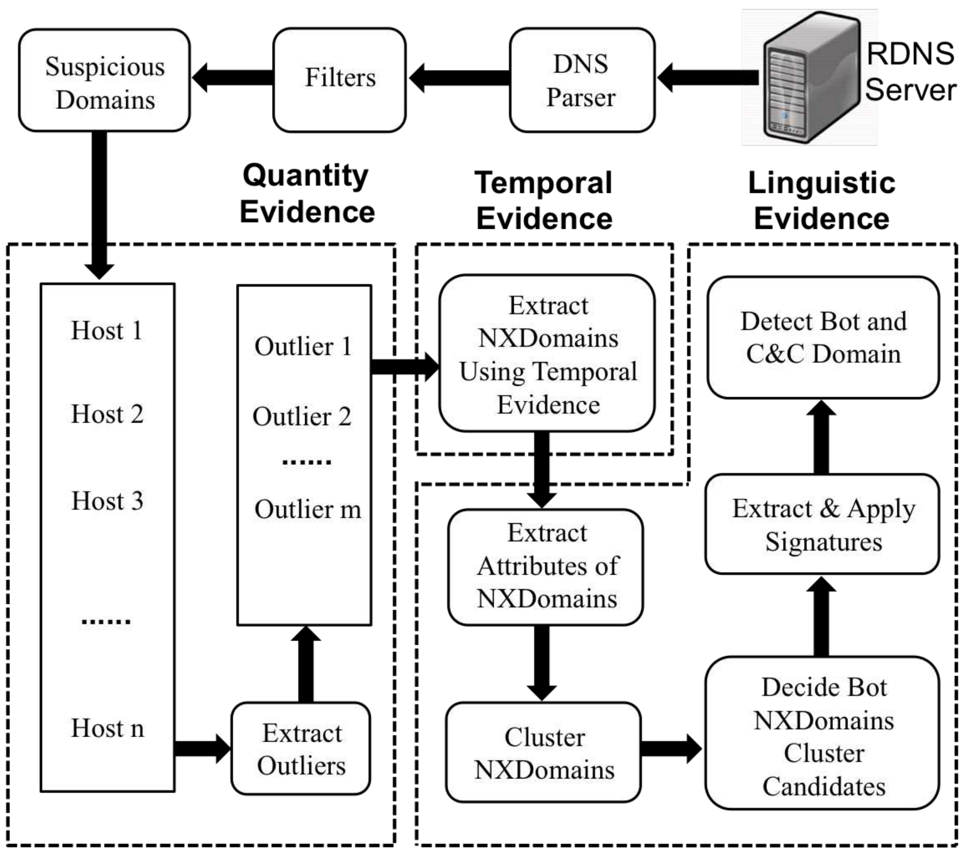

FIGURE 8.1. System Overview

8.1.3. ETHICAL CONSIDERATIONS. BotDigger does not require the IP address of the host making DNS requests, but only the domain request itself. This decouples the address of the host from the actual request; if there are many hosts in the network then it is hard to associate the request with the host. BotDigger still needs to report bots when detected, but this can be done using an opaque identifier for the host IP address, not the IP address itself. The identifier may be a mapping known only to the network operator and never revealed to BotDigger.

We acknowledge that in networks with a few hosts or in cases where the domain request itself contains host-specific information, privacy may still be compromised. However, we envision that BotDigger will be used the same way network operators use IDSs such as Snort and Suricata. Such IDSs require full access to packet headers and payload, and their use is justified as long as operators use them for network operations and security.

8.1.4. Filters. Previous work shows that many of the failed domains are non-malicious [47], [49]. Jiang et al. categorize failed DNS queries into seven groups in [47]. In [49], the authors classify NXDomains into nine groups. Based on their classifications, we build five filters to remove non-suspicious NXDomains.

(1) Overloaded DNS: Besides fetching the IP address of a domain, DNS queries are also "overloaded" to enable anti-spam and anti-virus techniques [70]. We collect 64 websites that provide blacklist services to filter overloaded DNS.
(2) Invalid TLD: We obtain the list of all the registered TLD from IANA [9]. A domain is considered as unsuspicious if its TLD is not registered.
(3) Typo of popular domains: We consider the top 1,000 domains in Alexa [2] and websites of world's biggest 500 companies from Forbes [8] as popular legitimate domains. If the *Levenshtein distance* between a given domain and any of these popular domains is less than a threshold, then the domain is considered as a typo.

(4) Excluded words: These domains contain the words that should not be included in queried domains, including ".local", "wpad.", "http://", and so forth.

(5) Repeated TLD: These domains could be introduced by misconfiguration of a web browser or other applications. An example is "www.example.com.example.com".

We use dataset CSUDNSTrace to study the effectiveness of each filter. Before applying the filters, we remove all the queried NXDomains (e.g., test.colostate.edu) under our university domain "colostate.edu". We list the number of filtered domains and their ratio to the total number of NXDomains in Table 8.1. From the table we can see that the filters can remove 87.3% of NXDomains, saving lots of computation resources and improving the performance.

TABLE 8.1. Filters Statistics

Filters	Filtered Domains	Percentage
Overloaded DNS	1232214	7.1%
Unregistered TLD	2488055	14.4%
Typo Domains	174515	1.01%
Misconfiguration words	7172856	41.4%
Repeated TLD	4046912	23.4%
All Filters	15114552	87.3%

8.1.5. QUANTITY EVIDENCE. Quantity evidence is based on the assumption that most hosts in a network are legitimate, and they do not query many suspicious 2LDNXs. On the other hand, a bot queries a lot of suspicious 2LDNXs. As a result, a bot can be considered as outlier in terms of the number of queried suspicious 2LDNXs. To define an outlier, we first calculate the average and standard deviation of the numbers of suspicious 2LDNXs for all the hosts, denoted as *avg* and *stddev*, respectively. Then we set the threshold as

$avg + 3 * stddev$. If the number of suspicious 2LDNXs queried by a host is greater than the threshold, then the host is labeled as an outlier.

We demonstrate quantity evidence using a real bot. First we pick one hour DNS trace from CSUDNSTrace and one bot trace from 140BotTraces. Then, we blend them together by adding the bot's DNS traffic on a randomly selected host in the background trace. Finally we have a one hour mixed DNS trace including 11,720 background hosts and a known bot. The CDF for the number of suspicious 2LDNXs queried by a host is shown in Figure 8.2. From the figure we can see that more than 95% hosts do not query any suspicious 2LDNX. On the contrary, the bot queries more than 20 suspicious 2LDNXs thus it is labeled as an outlier.

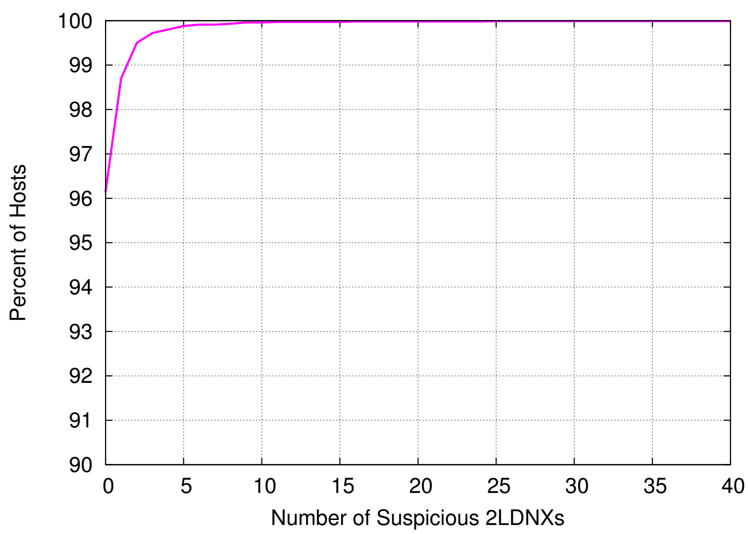

FIGURE 8.2. CDF for Suspicious 2LDNXs

8.1.6. TEMPORAL EVIDENCE. Most of the time, a bot behaves like legitimate host and it does not query many suspicious 2LDNXs. However, when the bot wants to look for the C&C domain, it will query many suspicious 2LDNXs. Consequently, the number of suspicious 2LDNXs suddenly increases. Then, once a bot hits the registered C&C domain, it will stop querying more DGA domains, thus the number of suspicious 2LDNXs will decrease. In other words, we are detecting a period of time when the number of suspicious 2LDNXs suddenly increases and decrease. This can be considered as a *Change Point Detection* (CPD) problem. We use *Cumulative Sum* (CUSUM) as the CPD algorithm because it has been proved to be effective and has been used in many other works, e.g., [92].

Let $X_n, n = 0, 1, 2, ...$ be the number of suspicious 2LDNXs queried by a given host every minute during a time window.

Let

$$\begin{cases} y_n = y_{n-1} + (X_n - \alpha)^+, \\ y_0 = 0, \end{cases} \tag{2}$$

where x^+ equals to x if $x > 0$ and 0 otherwise. α is the upper bound of suspicious 2LDNXs queried by legitimate host every minute. The basic idea of this algorithm is that $X_n - \alpha$ is negative when a host behaves normally, but it will suddenly increase to a relatively large positive number when a bot begins to query C&C domain.

Let b_N be the decision of whether a host has suddenly increased suspicious 2LDNXs at time n. A host is considered to have a sudden increase in suspicious 2LDNXs when b_N equals to 1. N is a threshold that indicates the number of suspicious 2LDNXs a host must reach before considered as bot candidate. N is specified by the user, as discussed in Section 8.2.1.

$$b_N(y_n) = \begin{cases} 0, & if \quad y_n \leq N \\ 1, & if \quad y_n > N \end{cases} \tag{3}$$

The method to detect a decrease in the number of suspicious 2LDs is similar to equation 2, and is defined in equation 4. The function to detect sudden decrease of suspicious 2LDNXs is the same as equation 3.

$$(4) \qquad \begin{cases} y_n = y_{n-1} + (X_{n-1} - X_n)^+, \\ y_0 = 0, \end{cases}$$

Now we investigate the temporal evidences using the same one hour DNS traffic blended in the last section. As it is hard to plot all the 11,720 hosts and due to the fact that most of them do not query many suspicious 2LDNXs, we only plot the suspicious 2LDNXs for 15 hosts including the bot in Figure 8.3. From the figure we can clearly see there is a spike appearing between minute 29 and 32, standing for the bot. However, we also notice that besides the bot, some legitimate hosts (e.g., host 10) also have spikes, thus they might also be labeled as suspicious. Spikes by legitimate hosts can result from user typos. Another explanation may be that each time Google Chrome starts it generates a number of failed DNS requests to determine if NXDomain rewriting is enabled [100]. We manually checked the NXDomains queried by the hosts that generated the spikes. While we found some of them not suspicious, we could not precisely pinpoint the reason for the spikes.

It is true that a single evidence is not strong enough to label a host as a bot. This is the reason why we use a chain of multiple evidences that helps to reduce false positives.

8.1.7. Linguistic Evidence. Linguistic evidence is built on two assumptions. The first is that the NXDomains queried by a bot are generated by the same algorithm, thus they share similar linguistic attributes (e.g., entropy, length of domains, etc.). On the contrary,

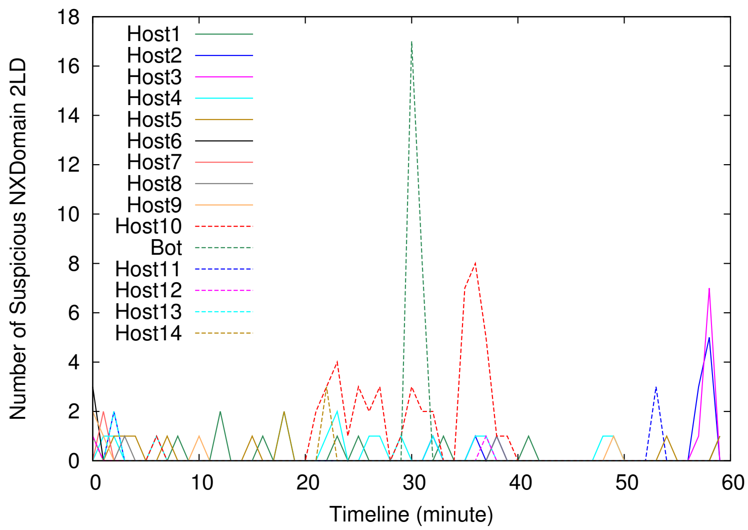

FIGURE 8.3. Timeline for Suspicious 2LDNXs

legitimate domains are not generated algorithmically but they are selected such that people can remember them easily. Consequently, the linguistic attributes of the NXDomains queried by a legitimate host are not similar to each other. The second assumption is that both DGA registered C&C domains and NXDomains are generated by the same algorithm. The only difference between them is whether the domain is registered or not. Consequently, C&C domains have similar linguistic attributes with DGA NXDomains. Based on these two assumptions, we first extract linguistic attributes from suspicious NXDomains and cluster the domains that have similar attributes together. After that, bot NXDomain cluster candidates are decided. Next, signatures are extracted from the cluster candidates and applied on successfully resolved domains looking for registered C&C domains.

8.1.7.1. *Linguistic Attributes.* We extend the attributes used in prior work [20], [76], [25], [18], [99] to 23 and list them in Table 8.2. From the table we can see that some attributes are dependent (e.g., length of dictionary words and percent of dictionary words). Currently we use all of these attributes, and we leave the study of attributes selection as a future work.

Table 8.2. Domain Linguistic Attributes

Index	Linguistic Attributes
1, 2	length of dictionary words in 2LD and 3LD
3, 4	percent of dictionary words in 2LD and 3LD
5, 6	length of the longest meaningful substring (LMS) in 2LD and 3LD
7, 8	percent of the length of the LMS in 2LD and 3LD
9, 10	entropy in 2LD and 3LD
11, 12	normalized entropy in 2LD and 3LD
13, 14	number of distinct digital characters in 2LD and 3LD
15, 16	percent of distinct digital characters in 2LD and 3LD
17, 18	number of distinct characters in 2LD and 3LD
19, 20	percent of distinct characters in 2LD and 3LD
21, 22	length of 2LD and 3LD
23	number of domain levels

8.1.7.2. *Dissimilarity Calculation.* Now we describe how to calculate dissimilarity of a single linguistic attribute between two domains. We denote two domains as D_1 and D_2, and denote their attributes as $a_{ij}, i = 1, 2$ and $1 \leq j \leq 23$. Dissimilarity of attribute j between D_1 and D_2 is denoted as $S_j(D_1, D_2)$ and calculated as follows.

$$S_j(D_1, D_2) = \begin{cases} 0 & \text{if } a_{1j} = 0 \, and \, a_{2j} = 0 \\ \frac{|a_{1j} - a_{2j}|}{max(a_{1j}, a_{2j})} & \text{else} \end{cases} \tag{5}$$

We use a modified version of Euclidean distance to calculate the overall dissimilarity of all the 23 attributes between two domains. Euclidean distance [32] has been used by others [18], [25]. The overall dissimilarity between D_1 and D_2 is denoted as $S_{All}(D_1, D_2)$ and calculated as equation 6. The smaller the dissimilarity, the more similar D_1 and D_2 are.

$$S_{All}(D_1, D_2) = \sqrt{\frac{\sum_{j=1}^{23} S_j(D_1, D_2)^2}{23}} \tag{6}$$

8.1.7.3. *Clustering NXDomains.* After extracting attributes from NXDomains and calculating dissimilarities, we run the single linkage hierarchical clustering algorithm to group the NXDomains that have similar attributes together. Process of the clustering algorithm is shown in Figure 8.4. Initially, each NXDomain denoted as orange dot is in a cluster of its own. Then, the clusters are combined into larger ones, until all NXDomains belong to the same cluster. At each step, the two clusters having the smallest dissimilarity are combined. The result of the clustering process can be depicted as a dendrogram, where a cut is used to separate the clusters, giving us a set of NXDomain clusters. For example, the dendrogram cut in Figure 8.4 gives two clusters. As we normalize the dissimilarities between domains, the dendrogram height is between 0 and 1. Currently the dendrogram cut height is determined experimentally, as shown in Section 8.2.1. As a future work, we plan to use statistical methods to cut the dendrogram dynamically. Finally, we compare the size of every cluster with a *BotClusterThreshold*. If a cluster has more NXDomains than the threshold, it is considered as a *bot NXDomain cluster candidate*, and the host is considered as a bot candidate.

8.1.7.4. *C&C Domain Detection.* After detecting bot NXDomain cluster candidates, we extract signatures from them, and then apply the signatures on successfully resolved domains to detect C&C domain.

For a given bot candidate c, all its bot NXDomain cluster candidates are denoted as $C_1, C_2, ...C_n$. We first combine these clusters as a $C = \bigcup_{i=1}^{n} C_i$. The NXDomains included in C are denoted as d_j. Each domain d_j contains 23 linguistic attributes a_{jk}, $1 \le k \le 23$.

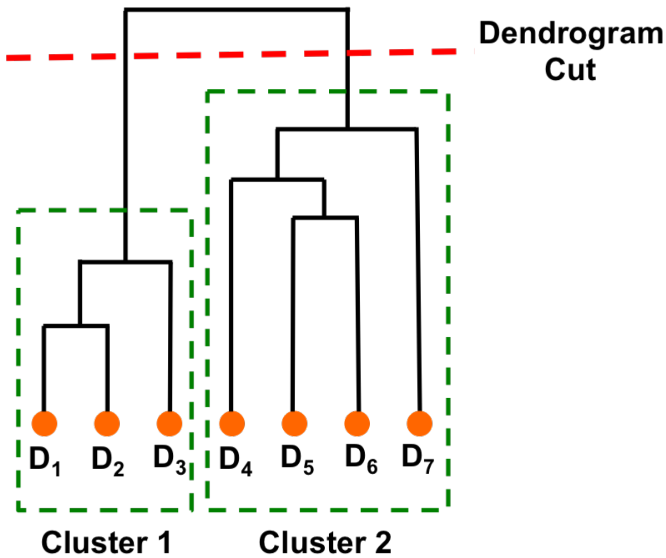

FIGURE 8.4. Hierarchical Clustering Dendrogram

Signatures are composed of an upper signature and a lower signature, each of them includes 23 values. The upper signature is denoted as $Sig_{upper} = (s_1, s_2, ...s_{23})$. Each value in the upper signature is defined as the maximum value of the corresponding linguistic attribute of all the domains in C, $s_k = max(a_{jk}), 1 \leq k \leq 23$, $d_j \in C$. Similarly, the lower signature, Sig_{lower} is defined as the minimum values.

Once we obtain the signatures, we apply them on the successfully resolved domains that are queried during (t_{begin}, t_{end}) to extract C&C domains. Recall that (t_{begin}, t_{end}) is decided by temporal evidence in Section 8.1.6. For a given successfully resolved domain, we first extract its 23 attributes. Then, for each attribute we check whether it falls within the corresponding attribute upper and lower bounds in the signature. Finally we compare the total number of matched attributes with a *SignatureThreshold*. If the former is greater, then we label the domain as C&C domain, and label the host as a bot.

8.2. Evaluation

8.2.1. True Positives. First, we evaluate the performance of BotDigger on detecting DGA-based bots. We use two botnets, Kraken and Conficker in our experiments. Recall that the dataset 140BotTraces includes 140 real Kraken traces that contain DNS queries/responses, C&C communication and other traffic. On the other hand, for the evaluation with the Conficker dataset we only have Conficker's DGA domains. We use these DGA domains to simulate 1000 bots. Each simulated bot randomly queries 20 domains from the Conficker domain pool every 10 seconds during the time window. We use 5 minutes as the time window in all of our experiments. At the end of every time window BotDigger analyzes the collected information and looks for bots. Users can decrease the time window if they want to detect bots more quickly, but decreasing the time window risks missing bots with slow activity.

Before running BotDigger on the two evaluation datasets we experimentally determine the parameters and thresholds introduced in Section 8.1. First, we use a one-day DNS trace from the CSUDNSTrace to decide α in equation 2. We find that more than 98% of the hosts query less than two suspicious 2LDNXs per minute, so we pick 2 as the α. N in equation 3 is set to the value of BotClusterThreshold. We then set SignatureThreshold experimentally by trying different values of the threshold and run BotDigger on the 1000 Conficker bots simulated above. In this specific experiment, we use 0.05 as the dendrogram cut. We will discuss how to pick the proper dendrogram cut in the next paragraph. The results are plotted in Figure 8.5. The x-axis and y-axis are bot detection rate and SignatureThreshold respectively and the different lines stand for different BotClusterThreshold. From the figure we can see that the bot detection rate is stable when the SignatureThreshold is less than 16, after that the rate drops quickly. As a result we set 16 as the SignatureThreshold.

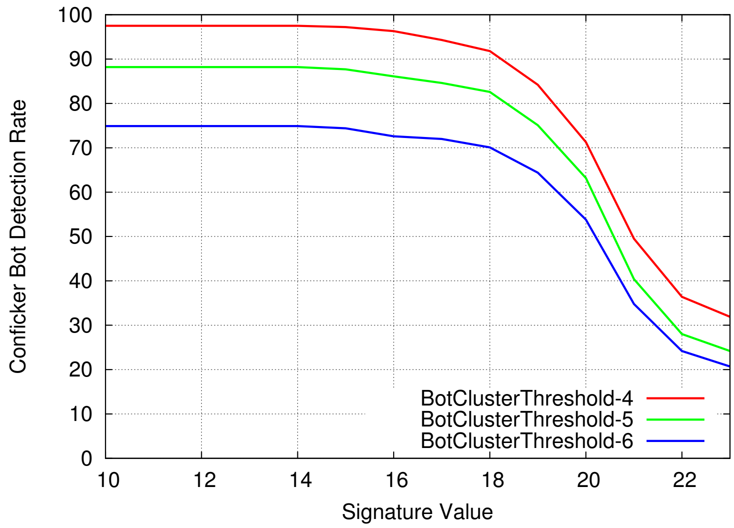

FIGURE 8.5. Signature Threshold Selection

Now we run BotDigger on Kraken and Conficker bots using combinations of two parameters, BotClusterThreshold and dendrogram cut in the hierarchical clustering algorithm. The results are shown in Figure 8.6 and Figure 8.7. The x-axis is the dendrogram cut, y-axis is the percentage of detected bots, and different lines stand for different BotClusterThreshold. From the figures we can see that by using 0.10 as the dendrogram cut, we are able to detect all the Kraken bots and 99.8% of Conficker bots.

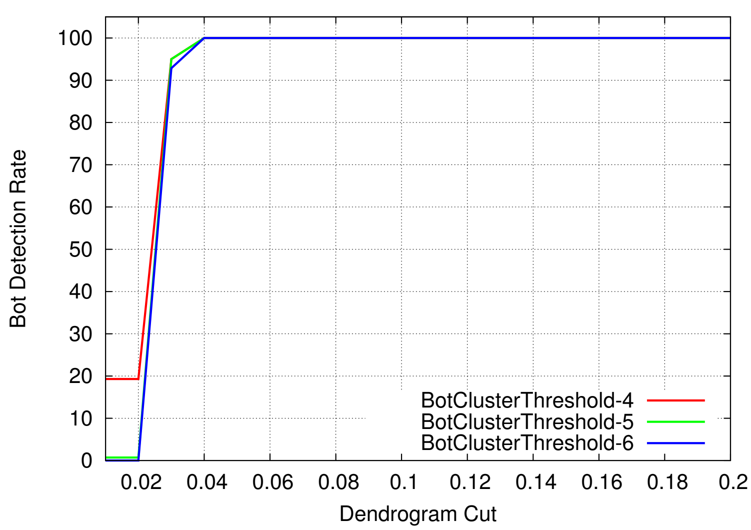

FIGURE 8.6. Kraken Bots Detection

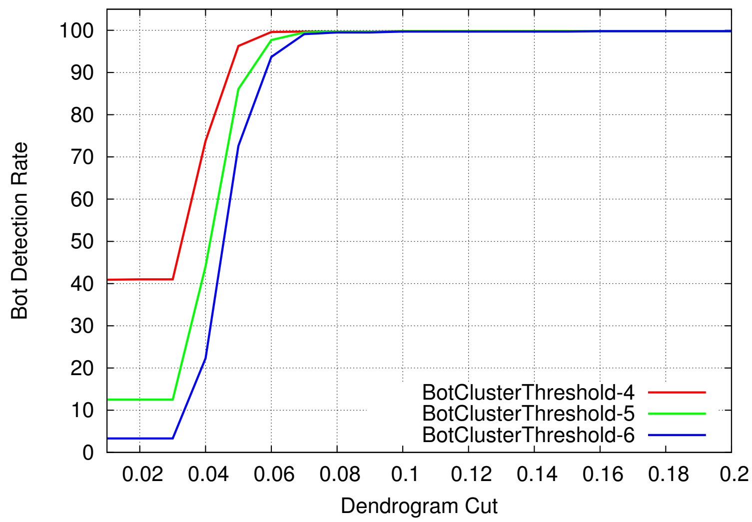

FIGURE 8.7. Conficker Bots Detection

8.2.2. FALSE POSITIVES. Besides DGA-based bots, some legitimate hosts could also query similar NXDomains that are clustered together and the host could falsely be labeled as a bot. We now evaluate such false positives by using the dataset CSUDNSTrace. As most of the users connected to CSU network are required to install anti-virus software, we assume CSUDNSTrace does not contain many bots. We use 0.10 as the dendrogram cut and 4 as BotClusterThreshold.

33 hosts (0.16% of all the hosts) are labeled as bots during the entire period of one week. We use two resources to check whether these domains and corresponding IPs are malicious. The first resource is *VirusTotal* [15], a website that provides virus, malware and a URL online scanning service based on 66 URL scanners, including ZeusTracker, Google Safebrowsing and Kaspersky among others. Another resource is *TrustedSource* by McAfee labs [11], a real-time reputation system that computes trustworthy scores for different Internet entities including IP addresses, domains and URLs. The scores are computed based on analyzing real-time global threat intelligence collected from McAfee's research centers'

centralized scanning systems and honeypots and from customers and end point software solutions around the world. We checked the labeled C&C domains and the corresponding IPs for each host using the above resources. If any of them is labeled as malicious or high risk, we consider the host as malicious. The results show that 22 hosts are labeled as malicious. In summary, we falsely label 11 hosts as bots, resulting in the false positive rate as 0.05%.

Within the 22 malicious hosts, we find that 5 hosts are highly suspicious. Most of the NXDomains queried by these hosts are random looking. In Figure 8.8, we list the NXDomains queried by such a host. After extracting signature and applying it on the successfully resolved domains, "sfaljyxfspbjftnv5.com" is labeled as C&C domain. The IP address of this C&C domain is 85.25.213.80. Upon further investigation on VirusTotal, we find that this IP is related to malicious activity and mapped to various random looking domains. Consequently, we believe this host is a bot.

obhgiunuht7f.com,	hlxgrygdmcpu8.com
ompxskwvcii3.com,	nnnwyujozrtnulqc5p.com
nwpofpjgzm6c.com,	dgpvsgsyeamuzfg2.com
kgapzmzekiowylxc5k.com,	nellwjfbdcfjl3g.com
rfsbkszgogjqlbm.com,	rcfwptxhgoiq27.com
unzxnzupscqxu.com,	okihwecmaftfxwz.com

Figure 8.8. NXDomains Queried by Suspicious Host A

Within the 11 false positives, we manually checked their queried NXDomains and found some interesting results. We include one example in Figure 8.9. From the figure we can find that some words appear frequently in the domains, such as "coach", "louis", "outlet", etc. In addition, we can also see that the domains contain typos. For example, "coach" and "louis" are misspelled as "coachh" and "llouis". One explanation of these typos is that the host tries to avoid conflicts with registered domains. In addition, we extract signature from NXDomains and apply it on the successfully resolved domains. By doing this, domain "www.golvlouisvuittonbags.com" is extracted. This domain is still in use, trying to sell replica Louis Vuitton. The above example shows that BotDigger is not only able to detect bots and suspicious hosts that query random looking domains, but is also capable of detecting hosts whose queried domains are generated from a set of dictionary words.

www.llouisvuittonoutlet.org,	www.iulouisvuittonoutlet.org
www.coachoutletsstoree.com,	www.illouisvuittonoutlet.com
www.coachfactorysoutlets.net,	www.louisvuittonof.com
www.coachfactoryonlines.net,	www.colvlouisvuittonoutlet.net
www.coachfactorystoreoutlete.org,	www.coachhoutlet.com

FIGURE 8.9. Domains Queried by Suspicious Host B

In addition to the CSUDNSTrace we also ran BotDigger on the traces captured in our research lab, Lab1, Lab2, and Lab3. Note that we do not regard these traces as ground truth. Although our network is well protected we cannot be certain it is bot-free. BotDigger detects four hosts as potential bots, denoted as $H_i, 1 \leq i \leq 4$. Then we try to confirm by running a bot detection system - BotHunter, on the three lab traces for independent verification. BotHunter labels H_1 as bot. Upon further investigation, we find that H_1 queries many domains and these domains are very similar as the ones queried by H_2. Moreover, some of the labeled C&C domains of H_1 are resolved to 80.69.67.46, which is the same IP as the labeled C&C domains for H_2. As a result, H_2 is very likely to be a bot. Besides, H_3 is highly suspicious because it queries many NXDomains, all of them beginning with "zzzzzzzzzzzgarbageisgreathere" (e.g., zzzzzzzzzzzgarbageisgreathere.funatic.cz, .penguin.se, inspirit.cz, etc). In summary, BotDigger introduces 1 false positive in three traces.

8.3. Discussion

A bot can bypass BotDigger by querying C&C domains very slowly, for example, querying a domain every 5 minutes. In this case, BotDigger may not detect it if a small time window is used. However, at least we make the bots less effective, meaning that it may take hours to contact to the C&C domains if the bots have to query tens of domains slowly. We can increase the time window to detect the bots that query domains slowly, but we expect more false positives will be introduced.

The quantity evidence in the evidence chain of BotDigger requires that the number of NXDomains queried by a bot is comparable more than legitimate hosts. As a results, BotDigger will fail to work if the bot is "lucky", meaning that it only queries a very small number of domains and hits the C&C domain. However, when the current C&C domain expires, the bot needs to look for the new C&C domain. It is very unlikely that the bot is "lucky" every time it looks for the C&C domain. Consequently, once the number of NXDomains queried by this bot matches the quantity evidence, BotDigger will analyze its DNS traffic for detection. In summary, a bot may evade the system for one time, but not all the time.

8.4. Summary

In this chapter, we introduce BotDigger, a system that detects DGA-based bots without a priori knowledge of the domain generation algorithm. A big advantage of BotDigger is that it can detect an individual bot by only analyzing DNS traffic collected from a single network. Any network administrator can run BotDigger without requiring additional information from other networks. A novel method - a chain of evidences, including quantity evidence, temporal evidence and linguistic evidence, is used in BotDigger for detection. We first use synthetic traffic to investigate each individual evidence and find many false positives. A chain of evidences helps reduce false positives because most of the legitimate hosts match one or two evidences but not all three. Two DGA-based botnets and two groups of background traces are used to evaluate BotDigger. The results show that BotDigger detects more than 99.8% of the bots with very low false positives.

CHAPTER 9

Conclusion and Future Work

9.1. Conclusion

In recent years, botnets began to use various evasion techniques to foil detection systems, including encrypted communications, dynamically generated C&C domains, and more. In this dissertation, we focus on detecting such advanced botnets in enterprise-like network environment.

The first challenge in detecting encrypted botnet traffic is that the lack of encrypted botnet traffic. Traces that are available are static, which prevents testing under various controlled scenarios. To address this problem we introduce BotTalker, a tool that can be used to generate customized encrypted botnet communication traffic. BotTalker includes a highly configurable encrypted-traffic converter along with real, non-encrypted bot traces and background traffic. BotTalker contributes to various communities, such as IDS developers, IDS customers, and IDS benchmark.

Armed with BotTalker, we generate two sets of encrypted botnet traffic and use them to evaluate the damage result from encrypted botnet traffic on three security systems. We find that encryption foils these systems greatly. In particular, encrypted botnet communications 1) make BotHunter(BH) miss 125 out of 140 bots in our botnet dataset; 2) suppress more than 33% of total alerts and more than 70% of unique alerts for Snort, and also make Snort miss 32% and 55% of bots in our two datasets; and 3) suppress more than 35% of total alerts and around 75% of unique alerts for Suricata, and also make Suricata miss 32% and 55% of bots in our two datasets.

After proving that encryption foils security systems, we introduce a method to detect encrypted botnet traffic based on the fact that encryption increases data's entropy. We introduce two high-entropy (HE) classifiers and add one of them to enhance BH by utilizing the other detectors it provides. Our HE classifier restores BH's ability to detect bots, even when they use encryption.

After that, we improve the HE classifiers by only looking at the first M packets of a flow to 1) improve the speed of bot detection tools, such as the enhanced version of BH, and 2) reduce the load on an IDS. The results show that by looking at the first 15 packets of each flow the traffic delivered to IDS is reduced by more than 50% while maintaining more than 99.9% of the original alerts. Comparing our traffic reduction scheme with other work we find that they need to inspect at least 13 times more packets than ours or they miss about 70 times of the alerts.

Finally, we introduce BotDigger, a system to detect DGA-based botnets. Previously, many systems have been introduced to detect DGA-based botnets. However, they suffer from several limitations, such as requiring DNS traffic collected across many networks, the presence of multiple bots from the same botnet, and so forth. These limitations make it very hard to detect individual bots when using traffic collected from a single network. The biggest advantage of BotDigger over the previous work is that it can detect single DGA-based bot by only using DNS traffic collected from a single network without a priori knowledge of the domain generation algorithm. We evaluate BotDigger's performance using traces from two DGA-based botnets: Kraken and Conflicker, and two groups of background traffic. Our results show that BotDigger detects all the Kraken bots and 99.8% of Conficker bots with very low false positives. Our results also show that BotDigger is not only able to detect bots and suspicious hosts that query random looking domains, but is also capable of detecting hosts whose queried domains are generated from a set of dictionary words.

9.2. Future Work

9.2.1. BotTalker.

BotTalker provides three encryption schemes, including packet level encryption, flow level encryption and SSL emulation. The timestamps for the former two remain the same, and we use four parameters to estimate the timestamps of emulated SSL connections in section 4.3.3. We plan to devise a more accurate method to estimate timestamps. For example, the key exchange and encryption algorithm negotiation phase and data transfer phase should use different timestamp estimation methods. Besides, currently the users have to use command line to configure the parameters and run BotTalker, we plan to provide a graphic user interface in the later implementation. Moreover, we will continue to add more bot traces and background traffic traces as they become available.

9.2.2. Traffic Reduction.

Our traffic reduction methodology sometimes marks opaque data as clear (e.g., 35% bzip files are misclassified as clear). This results in a slight reduction in efficiency, where we deliver opaque traffic to IDS. We could benefit from methods such as [94] that performs better in labeling opaque packets since they only focus on the characters whose ASCII values are less than 128. To improve our classifiers, we plan to use their method to label single packet as clear or opaque, and then use our early classifiers to label opaque flows.

9.2.3. BotDigger Sharing. BotDigger is able to detect bots within single network. We plan to build a platform where various organizations who run BotDigger can share bot information, for example, the DGA domain signatures. Assuming two organizations run BotDigger in their own networks. When a bot is detected at either side, a signature is published; when the other side detects a bot it also produces a signature, which can be compared with the other signature to determine if they came from the same bot family. In this case both sides detect bots, the new knowledge is if they are similar. This is useful to track the spread of bot infection.

Beyond signatures, there is more sharing that can be done, such as the domain names and IP addresses of the C&C servers, and all the other domain names that resolve to the same C&C IP address. This way other networks can pre-emptively block names and IP addresses detected elsewhere.

9.2.4. Put All Together. In this dissertation, we introduce several individual systems to detect botnet traffic and improve IDS performance. We plan to combine them together as a system to monitor all the traffic sent from Internet to the protected enterprise network, detects malicious traffic and infected clients in the enterprise network, and finally delivers the clean traffic. The overview of the system is shown in Figure 9.1. The system is composed of test module and detection module. In particular, we use BotTalker as the test module to evaluate any new encrypted botnet detection methods.

In the detection module, the traffic reduction first classifies all the incoming traffic into clear and opaque traffic. Then, the clear traffic and HE flow information go to encrypted botnet detection system (enhanced BotHunter) and DGA-based botnet detection system (BotDigger). BotDigger uses the DNS traffic to detect DGA-based botnets. The enhanced

BotHunter uses clear traffic and HE flow information to detect encrypted botnet traffic. The detected botnet traffic is dropped. The other traffic is delivered to IDS, where more malicious activities can be detected. Finally, the clean traffic is delivered to the enterprise network.

Note that the enhanced BotHunter uses HE flow information (whether a certain host has HE flows) for detection, so the encrypted botnet C&C traffic is not dropped here. Instead, the encrypted traffic is labeled as opaque and delivered to the enterprise network. To solve this problem, we can add another module to simply check whether the opaque traffic belongs to the detected bots. If so, the opaque will be dropped, otherwise, the traffic can be delivered to the network.

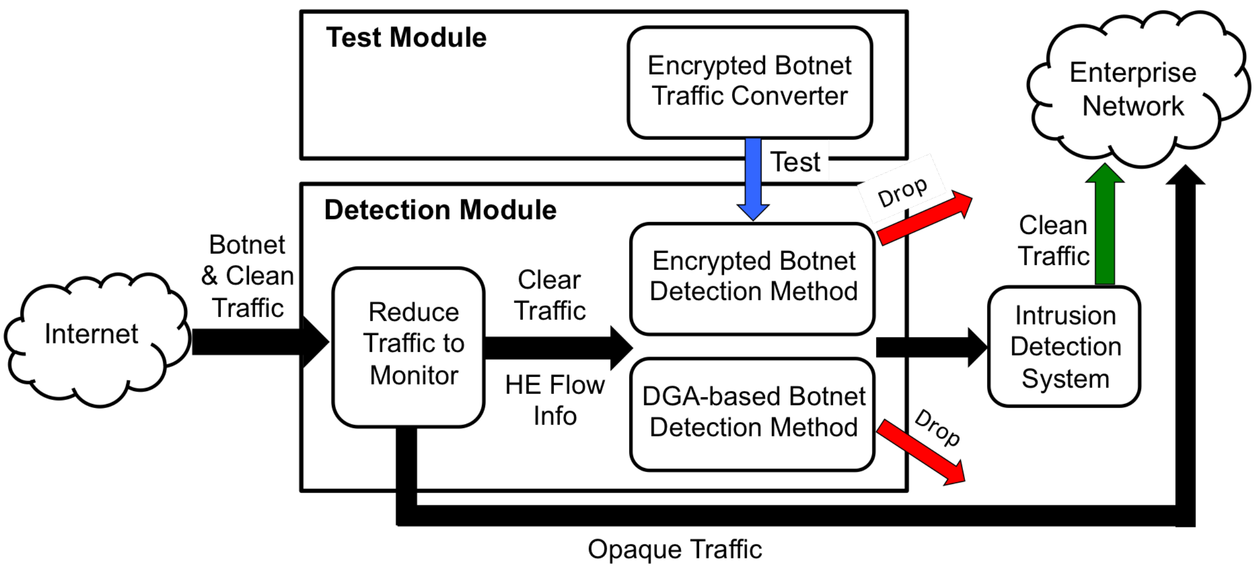

FIGURE 9.1. Advanced Botnet Detection System

Bibliography

[1] Advanced encryption package pro. http://www.aeppro.com.

[2] Alexa top sites. http://www.alexa.com.

[3] Bamital domains. http://noticeofpleadings.com.

[4] Bothunter dialog event. http://www.bothunter.net/about.html.

[5] Dyn. http://dyn.com.

[6] Emerging threats open ruleset. http://emergingthreats.net/open-source.

[7] Exeinfo pe. http://exeinfo.atwebpages.com.

[8] Forbes worlds biggest companies. http://www.forbes.com.

[9] Iana root zone database. http://www.iana.org/domains/root/db.

[10] Malware capture facility project. https://mcfp.agents.fel.cvut.cz.

[11] McAfee TrustedSource reputation system. http://trustedsource.org.

[12] The open source toolkit for ssl/tls. http://www.openssl.org.

[13] Snort. http://www.snort.org.

[14] Suricata. http://suricata-ids.org.

[15] Virustotal. https://www.virustotal.com/en/.

[16] Net losses: Estimating the global cost of cybercrime. Technical report, Intel Security Group (previously McAfee, Inc.), 2014.

[17] Symantec internet security threat report. Technical report, Symantec Corporation, 2016.

[18] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for dns. In *USENIX security symposium*, pages 273–290, 2010.

[19] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, II, and David Dagon. Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 27–27, Berkeley, CA, USA, 2011. USENIX Association.

[20] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou II, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX security symposium*, pages 491–506, 2012.

[21] Paul Barford and Mike Blodgett. Toward botnet mesocosms. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, Berkeley, CA, USA, 2007. USENIX Association.

[22] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '98/PERFORMANCE '98, pages 151–160, New York, NY, USA, 1998. ACM.

[23] Paul Barford and Vinod Yegneswaran. An inside look at botnets. In *Malware Detection*, volume 27 of *Advances in Information Security*, pages 171–191. Springer, 2007.

[24] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. A view on current malware behaviors. In *Proceedings of 2nd USENIX Conference on*

Large-scale Exploits and Emergent Threats, LEET, Berkeley, CA, USA, 2009. USENIX Association.

[25] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *NDSS*. The Internet Society, 2011.

[26] Arian Br, Antonio Paciello, and Peter Romirer-Maierhofer. Trapping botnets by dns failure graphs: Validation, extension and application to a 3g network. In *INFOCOM*, pages 3159–3164. IEEE, 2013.

[27] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[28] Marco Canini, Damien Fay, David J. Miller, Andrew W. Moore, and Raffaele Bolla. Per flow packet sampling for high-speed network monitoring. In *Proceedings of International Conference on Communication Systems and Networks*, COMSNETS, 2009.

[29] Ken Chiang and Levi Lloyd. A case study of the rustock rootkit and spam bot. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, pages 10–10, Berkeley, CA, USA, 2007. USENIX Association.

[30] Hyunsang Choi and Heejo Lee. Identifying botnets by capturing group activities in dns traffic. *Comput. Netw.*, 56(1):20–33, January 2012.

[31] Hyunsang Choi, Heejo Lee, and Hyogon Kim. Botgad: Detecting botnets by capturing group activities in network traffic. In *Proceedings of the Fourth International ICST*

Conference on COMmunication System softWAre and middlewaRE, COMSWARE '09, pages 2:1–2:8, New York, NY, USA, 2009. ACM.

[32] Michel Marie Deza and Elena Deza. *Encyclopedia of distances*. Springer, 2009.

[33] Christian J. Dietrich, Christian Rossow, Felix C. Freiling, Herbert Bos, Maarten van Steen, and Norbert Pohlmann. On Botnets that use DNS for Command and Control. In *Proceedings of European Conference on Computer Network Defense*, 2011.

[34] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings 15th ACM Conference on Computer and Communications Security*, pages 51–62, 2008.

[35] P. Dorfinger. Real-time detection of encrypted traffic based on entropy estimation. Masters thesis, Salzburg University of Applied Sciences, 2010.

[36] Peter Dorfinger, Georg Panholzer, and Wolfgang John. Entropy estimation for real-time encrypted traffic identification. In *Proceedings of the Third International Conference on Traffic Monitoring and Analysis*, TMA'11, pages 164–171, Berlin, Heidelberg, 2011. Springer-Verlag.

[37] Peter Dorfinger, Georg Panholzer, Brian Trammell, and Teresa Pepe. Entropy-based traffic filtering to support real-time skype detection. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, pages 747–751, 2010.

[38] Jan Goebel and Thorsten Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, Berkeley, CA, USA, 2007. USENIX Association.

[39] Jean Goubault-larrecq and Julien Olivain. Detecting subverted cryptographic protocols by entropy checking. 2006.

[40] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[41] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium*, SS'07, pages 12:1–12:16, Berkeley, CA, USA, 2007. USENIX Association.

[42] Guofei Gu, Vinod Yegneswaran, Phillip Porras, Jennifer Stoll, and Wenke Lee. Active botnet probing to identify obscure command and control channels. In *Proceedings of 2009 Annual Computer Security Applications Conference (ACSAC'09)*, December 2009.

[43] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[44] Joshua W Haines, Richard P Lippmann, David J Fried, MA Zissman, and E Tran. 1999 darpa intrusion detection evaluation: Design and procedures. Technical report, DTIC Document, 2001.

[45] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 9:1–9:9, Berkeley, CA, USA, 2008. USENIX Association.

[46] Alden W. Jackson, David Lapsley, Christine Jones, Mudge Zatko, Chaos Golubitsky, and W. Timothy Strayer. Slingbot: A system for live investigation of next generation botnets. In *Proceedings of Cybersecurity Applications & Technology Conference for Homeland Security*, CATCH '09, pages 313–318, Washington, DC, USA, 2009. IEEE Computer Society.

[47] Nan Jiang, Jin Cao, Yu Jin, Li Erran Li, and Zhi-Li Zhang. Identifying suspicious activities through dns failure graph analysis. In *Proceedings of the The 18th IEEE International Conference on Network Protocols*, ICNP '10, pages 144–153, Washington, DC, USA, 2010. IEEE Computer Society.

[48] Chris Kanich, Nicholas Weavery, Damon McCoy, Tristan Halvorson, Christian Kreibichy, Kirill Levchenko, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage. Show me the money: Characterizing spam-advertised revenue. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 15–15, Berkeley, CA, USA, 2011. USENIX Association.

[49] Yuta Kazato, Kensuke Fukuda, and Toshiharu Sugawara. Towards classification of dns erroneous queries. In *Proceedings of the 9th Asian Internet Engineering Conference*, pages 25–32. ACM, 2013.

[50] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 201–206, New York, NY, USA, 2004. ACM.

[51] Ronny T Lampert, Christoph Sommer, Gerhard Münz, and Falko Dressler. Vermont-a versatile monitoring toolkit for ipfix and psamp. In *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation, MonAM*, volume 6, 2006.

[52] Felix Leder and Tillmann Werner. Containing conficker - webpage and tools. http://iv.cs.uni-bonn.de/conficker.

[53] Felix Leder and Tillmann Werner. Know your enemy: Containing conficker (april 2009).

[54] Christopher Patrick Lee. *Framework for Botnet Emulation and Analysis*. PhD thesis, Atlanta, GA, USA, 2009. AAI3364236.

[55] Tobias Limmer and Falko Dressler. Flow-based front payload aggregation. In *Proceedings of IEEE 34th Conference on Local Computer Networks*, pages 1102–1109. IEEE, 2009.

[56] Tobias Limmer and Falko Dressler. Dialog-based payload aggregation for intrusion detection. In *Proceedings of the 17th ACM conference on Computer and Communications Security*, pages 708–710. ACM, 2010.

[57] Tobias Limmer and Falko Dressler. Improving the performance of intrusion detection using dialog-based payload aggregation. In *Proceedings of Global Internet Symposium*, GI '11, pages 822–827. IEEE Computer Society, 2011.

[58] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[59] Richard P Lippmann, David J Fried, Isaac Graf, Joshua W Haines, Kristopher R Kendall, David McClung, Dan Weber, Seth E Webster, Dan Wyschogrod, Robert K Cunningham, et al. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 2, pages 12–26. IEEE, 2000.

[60] Chen Lu and Richard R. Brooks. Timing analysis in p2p botnet traffic using probabilistic context-free grammars. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13, pages 14:1–14:4, New York, NY, USA, 2013. ACM.

[61] Robert Lyda and James Hamrock. Using entropy analysis to find encrypted and packed malware. *IEEE Security and Privacy*, 5(2):40–45, March 2007.

[62] Gregor Maier, Robin Sommer, Holger Dreger, Anja Feldmann, Vern Paxson, and Fabian Schneider. Enriching network security analysis with time travel. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*, SIGCOMM '08, pages 183–194, New York, NY, USA, 2008. ACM.

[63] P. Malhotra. Detection of encrypted streams for egress monitoring. Masters thesis, Iowa State University, 2007.

[64] Cédric Michel and Ludovic Mé. *ADeLe: An Attack Description Language for Knowledge-Based Intrusion Detection*, pages 353–368. Springer US, Boston, MA, 2001.

[65] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: finding p2p bots with structured graph analysis. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 7–7, Berkeley, CA, USA, 2010. USENIX Association.

[66] Kedar Namjoshi and Girija Narlikar. Robust and fast pattern matching for intrusion detection. In *Proceedings of INFOCOM*, pages 1–9. IEEE, 2010.

[67] Antonis Papadogiannakis, Michalis Polychronakis, and Evangelos P. Markatos. Improving the accuracy of network intrusion detection systems under load using selective

packet discarding. In *Proceedings of the Third European Workshop on System Security*, EUROSEC '10, pages 15–21, New York, NY, USA, 2010. ACM.

[68] Antonis Papadogiannakis, Michalis Polychronakis, and Evangelos P. Markatos. Tolerating overload attacks against packet capturing systems. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 197–202, Boston, MA, 2012. USENIX.

[69] Antonis Papadogiannakis, Michalis Polychronakis, and Evangelos P. Markatos. Scap: Stream-oriented network traffic capture and analysis for high-speed networks. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 441–454, New York, NY, USA, 2013. ACM.

[70] David Plonka and Paul Barford. Context-aware clustering of dns query traffic. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, pages 217–230, New York, NY, USA, 2008. ACM.

[71] Phillip Porras, Hassen Sadi, and Vinod Yegneswaran. A multi-perspective analysis of the storm (peacomm) worm. available at: http://www.cyber-ta.org/pubs/stormworm/report, 2007.

[72] Anirudh Ramachandran, Nick Feamster, and David Dagon. Revealing botnet membership using dnsbl counter-intelligence. In *Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet - Volume 2*, SRUTI'06, pages 8–8, Berkeley, CA, USA, 2006. USENIX Association.

[73] Konrad Rieck, Guido Schwenk, Tobias Limmer, Thorsten Holz, and Pavel Laskov. Botzilla: Detecting the "phoning home" of malicious software. In *Proceedings of ACM*

Symposium on Applied Computing, SAC '10, pages 1978–1984, New York, NY, USA, 2010. ACM.

[74] Christian Rossow and Christian J. Dietrich. Provex: Detecting botnets with encrypted command and control channels. In *Proceedings 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'13, pages 21–40, Berlin, Heidelberg, 2013. Springer-Verlag.

[75] P. Royal. Analysis of the kraken botnet, 2008.

[76] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. Phoenix: Dga-based botnet tracking and intelligence. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 192–211. Springer, 2014.

[77] Claude E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27:379–423, July 1948.

[78] S. Shevchenko. Domain name generator for murofet. http://blog.threatexpert.com/2010/10/domain-name-generator-for-murofet.html.

[79] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, pages 357–374, 2012.

[80] Greg Sinclair, Chris Nunnery, and BB-H Kang. The waledac protocol: The how and why. In *Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 69–77. IEEE, 2009.

[81] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 68–81, New York, NY, USA, 2004. ACM.

[82] Joel Sommers, Rhys Alistair Bowden, Brian Eriksson, Paul Barford, Matthew Roughan, and Nick G. Duffield. Efficient network-wide flow record generation. In *Proceedings of the 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*, pages 2363–2371. IEEE, 2011.

[83] Joel Sommers, Vinod Yegneswaran, and Paul Barford. A framework for malicious workload generation. In *Proceedings 4th Conference on Internet Measurement*, IMC '04, pages 82–87, New York, 2004. ACM.

[84] Anna Sperotto, Ramin Sadre, and Aiko Pras. *Anomaly Characterization in Flow-Based Traffic Time Series*, pages 15–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[85] Anna Sperotto, Ramin Sadre, Frank Vliet, and Aiko Pras. A labeled data set for flow-based intrusion detection. In *Proceedings 9th IEEE International Workshop on IP Operations and Management*, IPOM '09, pages 39–50, Berlin, Heidelberg, 2009. Springer-Verlag.

[86] Elizabeth Stinson and John C. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *Proceedings of the 2nd USENIX Workshop on Offensive Technologies*, pages 5:1–5:9, Berkeley, CA, USA, 2008. USENIX Association.

[87] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 635–647. ACM, 2009.

[88] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the storm and nugache trojans: P2P is here. In *;login*, 2007.

[89] W. Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. *Botnet Detection Based on Network Behavior*, pages 1–24. Springer US, Boston, MA, 2008.

[90] G. Stringhini, T. Holz, B. Stone-Gross, C. Kruegel, and G. Vigna. BotMagnifier: Locating Spambots on the Internet. In *Proceedings of the USENIX Security Symposium*, S. Francisco, CA, August 2011.

[91] Ricardo Villamarín-Salomón and José Carlos Brustoloni. Bayesian bot detection based on dns traffic similarity. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 2035–2041, New York, NY, USA, 2009. ACM.

[92] Haining Wang, Danlu Zhang, and Kang G Shin. Detecting syn flooding attacks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies.*, 2002.

[93] Ping Wang, Sherri Sparks, and Cliff C. Zou. An advanced hybrid peer-to-peer botnet. In *Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets*, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association.

[94] Andrew M. White, Srinivas Krishnan, Michael Bailey, Fabian Monrose, and Phillip Porras. Clear and present data: Opaque traffic and its security implications for the future. In *Proceedings of the Network and Distributed Systems Security Symposium*, 2013.

[95] Julia Wolf. Technical details of srizbis domain generation algorithm. https://www.fireeye.com/blog/threat-research/2008/11/technical-details-of-srizbis-domain-generation-algorithm.html.

[96] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In *Proceedings of*

the 14th European conference on Research in computer security, ESORICS'09, pages 232–249, Berlin, Heidelberg, 2009. Springer-Verlag.

[97] Sandeep Yadav and A. L. Narasimha Reddy. Winning with dns failures: Strategies for faster botnet detection. In *SecureComm*, volume 96 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 446–459. Springer, 2011.

[98] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A. L. Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated domain-flux attacks with dns traffic analysis. *IEEE/ACM Trans. Netw.*, 20(5):1663–1677, October 2012.

[99] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, pages 48–61. ACM, 2010.

[100] Fyodor Yarochkin, Vladimir Kropotov, Yennun Huang, Guo-Kai Ni, Sy-Yen Kuo, and Yi Chen. Investigating dns traffic anomalies for malicious activities. In *Proceedings of the 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 1–7. IEEE, 2013.

[101] Ting-Fang Yen and Michael K. Reiter. Are your hosts trading or plotting? telling p2p file-sharing and bots apart. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, ICDCS '10, pages 241–252, Washington, DC, USA, 2010. IEEE Computer Society.