The Pennsylvania State University
The Graduate School

SALTING PUBLIC TRACES WITH ATTACK TRAFFIC TO TEST FLOW CLASSIFIERS

A Thesis in
Computer Science and Engineering
by
Zeynel Berkay Celik

© 2011 Zeynel Berkay Celik

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2011

The thesis of Zeynel Berkay Celik was reviewed and approved* by the following:

George Kesidis
Professor of Computer Science and Engineering
Thesis Advisor

David J. Miller
Professor of Electrical Engineering

Sencun Zhu
Associate Professor of Computer Science and Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

We consider the problem of using flow-level data for detection of botnet command and control (C&C) activity. We find that current approaches do not consider timing-based calibration of the C&C traffic traces prior to using this traffic to salt a background traffic trace. Thus, timing-based features of the C&C traffic may be artificially distinctive, potentially leading to (unrealistically) optimistic flow classification results.

In this thesis, we show that round-trip times (RTT) of the C&C traffic are significantly smaller than that of the background traffic. We present a method to calibrate the timing-based features of the simulated botnet traffic by estimating eligible RTT samples from the background traffic. We then salt C&C traffic, and design flow classifiers under four scenarios: with and without calibrating timing-based features of C&C traffic, without using timing-based features, and calibrating C&C traffic only in the test set. In the flow classifier, we strive to use features that are not readily susceptible to obfuscation or tampering such as port numbers or protocol-specific information in the payload header. We discuss the results for several supervised classifiers, evaluating botnet C&C traffic precision, recall, and overall classification accuracy. Our experiments reveal to what extent the presence of timing artifacts in botnet traces leads to changes in classifier results, and we show that the presence of timing artifacts in botnet traces can lead to changes in classifier/network intrusion detection system (NIDS) results.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I offer my sincerest gratitude to my advisor, Dr. George Kesidis, who has given me the constant motivation and support to pursue this research. I attribute the level of my masters degree to his encouragement and effort. I am also very grateful to Dr. David J. Miller for his guidance and providing his input throughout my master's thesis research. My sincere thanks to Dr. Sencun Zhu for granting me his precious time. I would also like to extend my thanks to my colleagues, Jayaram Raghuram and Xiaonan Zang for bearing with me during our technical discussion. Many thanks go in particular to Umut Orhan and Zikri Bayraktar. I am much indebted to Umut for his valuable advice in introducing me to new programming skills. I have also benefited by advice and guidance from Zikri who also always kindly grants me his time even for some of my unintelligent questions.

Dedication

For my grandpa,
Melek, “Angel”,
I was not around, when you had moved to Heaven...

Chapter 1

Introduction

1.1 Botnets and Thesis Statement

The motives of Internet attackers have shifted from the pursuit of thrills to mercenary goals, political pressure and espionage, and they have become more organized and professional in the last few years. As a result, malicious software has become relentlessly inventive, and is presented on a variety of platforms; from servers to mobile phones. These malwares constantly evolve to find new ways to infect, spread and achieve their malicious goals such as stealing sensitive data, shutting down the systems of enterprise infrastructure, and last but not least, distributing various forms of content including e-mail spam. It is expected that the evolving nature of computing will facilitate the continued problems in the era on Internet security.

Botnets have been on the Internet for over 10 years and are perceived as one of the most serious security threats today. They construct a network of compromised computers called "bots" (or zombies), under the remote control of a botmaster. Botmasters control bots as a platform to carry out different malicious activities such as sending e-mail spam, performing distributed denial of service (DDoS) attacks, click frauds, and data exfiltration. Botnets have evolved over time to become stealthier and more resilient, which make them difficult to detect and shut down. Recently, a FBI investigation stated that botnets have infected over one million hosts, and caused economic loss of more than $20 million. It is also estimated that every year these damages will increase [6].

For over a decade, various types of botnets have been developed. Initially, IRC (Internet Relay Chat) based botnets coordinated by a single botmaster through an IRC C&C channel were predominant. However, as botnet detection techniques for IRC bot-

nets improved, botnet infection, communication and propagation forms evolved. The characteristic of C&C channel has evolved from IRC to HTTP, POP, and the structure of communication patterns has shifted from centralized to more sophisticated and decentralized structures such as peer-to-peer (P2P), and where fast flux DNS (Domain Name System) services are commonplace.

Despite the further development of various botnet detection techniques, tracing the origin and detecting is in its infancy. Both host-based solutions to network-based solutions, whether they use statistical anomaly or signature based techniques, have been adapted to prevent and mitigate botnets. These techniques differ in the type of data they are applying. One may employ flow-based features, DNS logs, or packet data features (we refer these techniques as flow-based, DNS-based, and packet-based respectively).

Packet-based approaches check for known strings or regular expressions in the byte level signatures to all headers up to the application layer (complete packet payload)[50]. After the emergence of fast-flux service networks (FFSN)[8], which are used to increase resilience and anonymity of botnets, DNS-based approaches have also evolved. DNS-based approaches mainly use either DNS query/response to detect the hosts behaving as a reverse proxy and to identify the motherships, or use access patterns of the infected hosts to the C&C servers. Flow-based detection relies on information gathered from multiple packet headers of a single defined flow and its associated features which provides a view of network traffic by tracking session flows between multiple endpoints simultaneously. Flow-based intrusion detection analyzes aggregated network traffic statistics to detect botnets.

Detection based purely on flow-based features, particularly using timing information, is the focus of this thesis. However, we consider neither any specific behavioral signatures that may be given nor use timing information to detect anomalies that may indicate communication by stepping stones. Rather, our objective is to study detection of botnet C&C activity by flow-based classification.

Botnet detection techniques were adapted performing machine learning (ML) algorithms use timing-based flow statistics (*e.g.*, packet or byte rate, inter-packet spacing, and flow duration)[22, 38, 56]. These techniques use representative attack traffic for both training and testing their classifiers to provide confidence in their proposed solutions. However, there are not many realistic datasets publicly available with background and attack traffic naturally present together. For these reasons, researchers often generate synthetic malicious traffic to salt real enterprise packet traces for evaluating the effectiveness of the proposed classifiers.

The research presented in this thesis focuses primarily on exploration of how the timing-based features determined during the salting process may affect classification performance. Timing-based features may vary depending on the position of the packet-trace monitor and, of course, on the geographical separation of communicating points including bots and their masters (and intermediate stepping stones if any). In both situations, "directly" salting botnet traces into the background traffic may introduce timing inconsistencies with the background traffic which may affect classification performance. Hence, it is necessary to achieve timing consistency between the traces to mitigate specific artifacts of timing-based features.

Our objective is, before salting simulated botnet traffic traces into background traffic traces, to provide consistency between the timing-based features of the two traces. For these reasons, we analyze the factors that contribute to the timing-based features of flow classifiers. After simulating a benign version of the Kaiten bot to acquire botnet C&C traces, we estimate eligible round-trip time (RTT) samples from the background traffic. We calibrate the simulated botnet traffic timestamps to provide consistency of timing-based features between the traces. Then, background traffic is salted to the simulated botnet traces, and flow features are extracted. We perform experiments using supervised ML algorithms to distinguish the botnet C&C traffic from the background traffic, and discuss the classification results both with and without calibrating timing-based features of botnet C&C traffic. Our results shows that the presence of any timing artifacts in botnet traces leads to changes in classifier results for some classifier models, and this could result in a serious impact on detection and false negative rates of classifier/network intrusion detection system (NIDS) results.

1.2 Outline

This work is outlined as follows: in Chapter 2, we provide an in-depth description of the botnet taxonomy and evolution. This includes an exploration of the botnet developments with given examples. We follow with Chapter 3, which covers the state of the art detection approaches including packet-based, DNS-based, and flow-based approaches. While other chapters particularly focus on flow-based detection approaches, Chapter 3 offers a general description of botnet detection techniques. In Chapter 4, we explain the IP flow generator framework design, including definitions from flow associated features, which are necessary to extract features from raw PCAP files in order to develop a flow classifier. Chapter 5 provides a description of the experimental set up. Through the

feature extraction from the background traffic, the way ground-truth flow-class labels derivation and IRC-based botnet Kaiten simulation. We next carefully analyzed the RTTs of the background and botnet C&C traffic to determine whether or not timing inconsistencies are possible. After analyzing the background and botnet traffic RTT samples, we determine that there is a difference between the mean values of the two traces which changes the timing-based features of the C&C traffic; our efforts then turn to a botnet RTT calibration and timestamp modification procedure to achieve timing consistency between two traces. Chapter 6 presents supervised ML algorithms used in our experiments. By bringing all of pieces together, in Chapter 7 we extract the flow features of the botnet traffic with the designed flow exporter system for different 4 scenarios. In the first scenario, the timing-based features of the botnet C&C traffic are not calibrated with the background traffic in both training and test set, in the second scenario they are calibrated with the background traffic in the training and test set, in the third scenario, the timing-based features are not used for classification. Finally, in the fourth scenario the timing-based features of the C&C traffic are not calibrated in the training set but are calibrated in the test set. We then applied four different supervised ML algorithms each with a different statistical background. These algorithms are suitable for multiclass problems and are commonly applied to ML problems in intrusion detection systems to observe the effects of the timing-based features in classifier performance. We explore the classification results of four different scenarios by presenting false positive, false negative rates of the C&C traffic class, and overall accuracy of the classifiers. Finally, Chapter 8 concludes the thesis and discusses the future of research in this field.

Chapter 2

Background

The increase in sophistication of the communication patterns for C&C challenges the normal approaches for detection and mitigation of the botnets. Detection of the botnets relies on understanding the architecture of the models and its underlying mechanisms. This section will cover botnet lifecycle spanning from infection stage to malicious activities. In addition, the following subsections shed light on comprehensive coverage of botnet evolution by categorizing of botnets depending on their attack behavior, C&C communication protocols, C&C rallying mechanisms and their commanding structure.

2.1 Botnet Taxonomy and Evolution

A general lifecycle of a botnet has commonly three stages: infection, maintenance and update, and malicious activities [21]. Multiple attack vectors are used to infect computers and spread bots. As an example, a malicious program can exploit a new or known unpatched vulnerability in the system, *e.g.*, a buffer overflow vulnerability in Microsoft Windows remote procedure call (RPC) and local security authority (LSA) services togerther with a buffer overflow error in the core Microsoft Windows dynamic-link library (DLL) has been successfully exploited by attackers [45]. Other examples of the infection include: tricking the user into downloading and executing malware (bot) code while viewing web pages or e-mail attachments, exchanging malicious content using P2P systems, and backdoors left by covert worms or remote access trojans. After systems are infected, bots are ready to connect to some special hosts (*e.g.*, C&C servers), and receive commands from their botmaster via defined hard-coded malware code. A bot may contact to a public server which the botmaster controls to issue commands to all infected

bots to construct coordinated attacks. During this stage, a bot may request updates from the C&C server or botmaster may perform several initial actions for maintenance. This might include downloading a more up-to-date version of the malware, testing the bandwidth of its connection, obtaining a list of C&C server names, or declaring its registration to the botnet. Bots are used to serve as automated agents to carry out different malicious activities such as sending spam, performing DDoS attacks, click frauds, data exfiltration, and phishing.

Distinctive Features	**Examples**
C&C structure	Centralized, P2P, Distributed, Fast-Flux network service
C&C communication protocol	IRC, HTTP, P2P
C&C rally mechanisms	Hard-coded IP addresses, dynamic DNS
Evasion techniques	Traffic Encryption, Rootkits
Malicious activity	DDOS, spamming, phishing, click fraud, data exfiltration

Table 2.1. Evolution of Botnet Architecture

IRC servers are the simplest rallying hosts that enable simple and low latency virtual communication between bots and botmasters. For this reason, IRC-based botnets with centralized C&C structures by utilizing the IRC are by far the preferred and most widely employed C&C communication models. In centralized C&C structures, upon infection the bot connects to the rallying host (rendezvous point, typically IRC server) using a predetermined port with a unique generated nickname and password, and waits for further commands from botmaster (known as PUSH mechanism or command forwarding, *e.g.*, used by Phatbot and Spybot)[22]. A rallying host is another compromised host through which the bostmaster controls the bots by issuing them commands, thus forming a centralized C&C architecture.

Another mechanism to spread the commands of the bosmaster is the PULL (command publishing/subscribing) technique [24]. In this technique, the botmaster uploads a command file on the rallying host (*e.g.*, web server). The bots periodically or frequently connect to the rallying host and download the files for the attacks. In this type of botnet, an HTTP server is used for communication model. HTTP-based C&C is still centralized as IRC-based bots; however HTTP protocol differs from the IRC protocol where the C&C traffic blends into normal HTTP traffic. The benefits of using HTTP as a C&C channel are twofold: first, since the IRC protocol for botnets is well-known, existing intrusion detection systems (IDS) can carefully examine suspected IRC traffic to detect and prevent botnets. Thus, the HTTP protocol mixes botnet C&C traffic together with the majority of Internet traffic web applications making botnet harder to detect by the

IDS. Secondly, firewall policies may block incoming/outgoing IRC traffic of IRC ports to prevent joining or communicating with the rallying host. However, using HTTP as a C&C communication channel can usually bypass firewall security policies.

P2P botnets (*e.g.*, Phatbot, AgoBot, Nugache and Peacomm [17]) are the next generation in the evolution of botnets to overcome the the limitation of traditional centralized botnets. P2P botnets organize themselves in a decentralized way. A decentralized C&C structure does not offer a possible central point of failure for the botnets. Additionally, botmasters save money by not purchasing C&C DNS domains. In a P2P botnet, bots find and join a P2P network by connecting to several infected machine (a bootstrap procedure), rather than to centralized C&C server either using a predetermined list of peers or the locations of predetermined IP addresses that are hard-coded in the bot code [62]. The botmaster defines a set of commands in the P2P system, and launches the commands by publishing a command; then all nodes transfers the received command to other bots using their neighbor (buddy) lists. In order to construct these lists, P2P bots have to communicate constantly with their neighbors to receive commands and have to send KEEP ALIVE messages to other bots in the network [18] which makes coordination of bots difficult compared to the centralized structure.

After bots utilize the traffic protocols to communicate with C&C center or other bots, another issue emerges with specifying the IP addresses of the rallying hosts for commanding the bots. Mostly, the IP addresses of the centralized rallying hosts are hard-coded in the binary code of the bot with *static IPs*. For that reason, shutting down the rallying host would bring down the entire bot infrastructure (*e.g.*, centralized IRC botnets with one C&C control server). To make the infrastructure more robust, malware authors use "bot-herding" [16]. In this approach, the location of rallying host(s) is dynamically set in order to keep the rallying hosts portable. The botmaster sets a list of IP addresses in bot code, and changes the IP addresses of the rallying hosts periodically using the dynamic DNS (DDNS). Each time a rallying host is shut down, the botmaster would set up new rallying host(s) and update the appropriate dynamic DNS entry. Bots migrate to their new rallying host either by commands given by the botmaster or with the maintenance/update commands of the software [13]. Another approach to locate the rallying host is to use the domain generation algorithm (DGA) [55]. The list of the rallying hosts are computed from the predefined algorithms embedded to binary code of the bots independently until rallying host provides a response (*e.g.*, Torpig uses DGA by seeding with the current date and a numerical parameter).

Apart from using DDNS to generate various rallying hosts by hard-coded or updated

IP addresses, another role of the DNS applied by the botmasters is using the short time-to-live (TTL) option of the DNS servers to cache the DNS records for short periods. [42] explains DNS as follows: When a client requests a web address, the browsers sends a request to designated (authoritive) DNS to translate domain names into IP addresses. DNS structure is a distributed database operating in a hierarchical data schema in which each server is responsible of name resolution in their zones. After DNS server gets the request, it checks its database entries to solve the requested domain name's IP address (*i.e.*, Address record (A-record)), contains hostname, IP address, TTL, class and type of the DNS record), and sends back to the client. If a DNS server cannot find entry, it queries its parent server for the IP address of particular domain name. This request continues until it reaches the authoritative server for the requested domain. At this point, to reduce the requesting delay of a web address each time and load of the DNS server, the designated DNS server caches the DNS resolution for a given TTL period of time after getting the successful domain name resolution from the requested web addresses DNS server. TTL specifies the local caching period of the DNS servers which is defined by the domain owner by modifying the DNS record of the domain name's zone file.

Fast-flux service network refers to a practice of continuously updating the DNS entries at regular intervals (*i.e.*, fast IP changing hosts). This procedure adds an extra layer to the botnet communication structure to increase resilience and anonymity of botnets. Instead of directly making requests to the web servers (motherships used for malicious activities); users access web pages using intermediary nodes (bots, agents) by allowing the domain names' multiple IP addresses. The implementation of the DNS allows the botmaster to utilize round robin DNS (RRDNS) and short TTLs. The TTL values of the DNS records are kept short in order to change the IP addresses of the compromised or bullet-proof web servers (servers which are rented without terms of service or acceptable use policies) used for malicious activities (particularly phishing and malware hosting operations [44]). As an example, when a victim clicks for a web page inside a spam e-mail, the authoritative server returns the one particular IP address among the multiple IP addresses of a domain name as a result of RRDNS. Actually, the IP address is obtained from the name server under control of the botmaster (*e.g.*, bullet-proof servers offered from loosely controlled countries), who is responsible for the name resolution of the malicious web page. The victim initiates a connection through the received IP address which belongs to a bot. After the bot gets the request from victim, it will redirect the request to the hosted web server (*i.e.*, behaving as a reverse proxy), and all responses will be from the unauthorized or illegal content host (mothership). In addition, motherships

are used as central servers by bots to get the updates or dowload plug-ins. In this way, botmasters prevent shutting down the services of the mothership by making impractical to trace-back the mothership. Because the received IP addresses from the authoritative DNS servers are proportional to botnet size, shutting down of the all domains obtained as a result of DNS query becomes impractical. The honeynet project [8] splits the fast flux network services into two categories: single-flux and double-flux service networks. In single-flux service networks, the DNS server (returns the requested IP addresses of the mothership) can be identified by tracking the DNS requests. However, in double-flux service networks even the IP addresses of the DNS nameservers change by using bots to proxy DNS traffic.

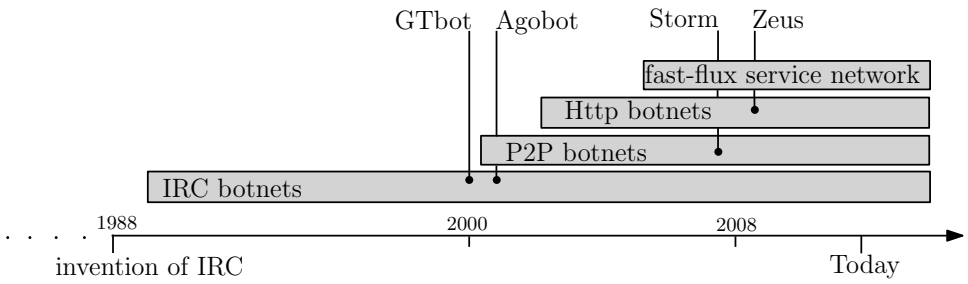

Figure 2.1. Timeline of Malicious Botnet Evolution

With the definitions given above, we now give examples from the most nefarious and commonly used botnet families. As shown in Figure 2.1, the evalution of the botnets started with the invention of the IRC, and first benevolent IRC-based bot GM appeared in 1989. The first notable nefarious botnet GTbot (global threat bot) appeared in 2000. GTbot was based on centralized C&C structure and IRC communication protocol. It infected users by downloading malware masquerading as registry cleaner software. Upon infection, the underlying program started, and connected the victim to the IRC channel. It was used for DDoS attacks to the online IRC users. Agobot (also called Phatbot or Gaobot) was written with a tidy and abstract C++ code with a modular design. Its C&C structure was also IRC-based, however it also used the WASTE application which was a decentralized chat, instant messaging and file sharing program and protocol [9]. Furthermore, it included functions to prevent reverse engineering by detecting debuggers and virtual machines, so a virtual machine-based honeypot could not run Agobot [67]. After 2002, botnets had a notable evolution with the development of more complex architectures due to open-source code exchange, free available botnet code samples, and user friendly interfaces. Additionally decentralized C&C centers, and HTTP-based C&C

communication channels have emerged.

In 2007, the first widespread P2P botnet Storm was released. Moreover, fast-flux networks have received wider attention, after it is employed by botnets. Storm Trojan infects victims through e-mail attachments. After infection, the victim joins Overnet P2P network based on Kademlia algorithm. Each victim has an identifier of a peer hash in order to provide a distributed hash tables to form decentralized architecture [19]. Then, the victim starts searching for a total of 146 peers that are hard-coded in the binary of the bot, and becomes a part of the network. The victim also searches for the secondary injection with a hash of a number and data combination. The response consists of URL that can be only decrypted with a hard-coded 64-bit RSA key. After secondary injection, the bot starts malicious activities such as sending spam e-mails, and participating in DDoS attacks. The botmaster dynamically sets the domain names for the update process which are fast-flux domains, and the fast-flux domains' TTL values set to very low (typically 0 to prevent caching in DNS servers) [44].

The Zeus botnet is first identified in 2007, and in 2009 it became more widespread. [12] yielded several insights about Zeus. It commonly infects users' system via spam emails which consist of links of malicious websites, and it lures users to download a particular Zeus botnet file. It does not have a built-in capability to spread to other computers by scanning vulnerabilities. After the infection stage, victim downloads a configuration file which contains a list of web banking websites to steal the users' credentials. When users visit a web page in the list, the Zeus botnet saves the keystrokes by activating the keylogger function, and saves all the keystrokes of the user. Zeus also has other malicious behaviors which can be updated from a rallying host, *e.g.*, injecting false fields into web forms which the user will unknowingly fill so that they submit more information than required by the web pages. At periodic intervals, the botnet connects to the web server and submits the saved files. All data is sent to the C&C center (*i.e.*, web server) encrypted by the RC4 algorithm in which a key is generated by using bot specific password. The web servers include bulletproof, hacked, and free hosting. They change by using FFSN and run on an IIS or Apache web server with MySQL implantation to provide communication between bots and botmaster. In addition, Zeus saves all data to the database that is ready for sharing with its customers.

Chapter 3

Botnet Detection

Defending against botnets is an extremely challenging problem. Along with the continuous evolution of botnets, bot coders find different ways to evade proposed solutions, and continue their malicious attacks. This results in a never ending arms race between the attackers and defenders. Researchers have proposed several approaches to detect botnets and their operators. Apart from the detection methods, there are techniques utilized by the botnet detection such as botnet tracking and botnet size measurement. As an example, honeynets [8] are used to learn botnet characteristics, targets and tools used by the botmaster when compromising a system. Sinkholing (*e.g.*, taking down the malicious server to take control of the Torpig botnet [55]) are used for estimating size of the infected host and malicious behaviours of the botnets. These techniques are commonly used in detection methods in order to collect botnet traces, discover botnet strategies and hijack C&C servers to understand the organizational nature and potential threat of botnet.

Botnet detection methods can be categorized either as *misuse (signature)* or statistical *anomaly* detection depending on their implementation. Misuse detection relies on a signature or pattern database. The signatures are generated by analyzing the possible or used practices and strategies of botnets. As an example, a signature can be used for checking whether particular nicknames of botnets or common C&C center IP addresses from the blacklists present in packet payload. Anomaly detection defines behaviors or profiles of hosts and network, and any behavior is accepted as an intrusion that falls outside the predefined model of the behavior. As an example, techniques checking per IP address query rates applied to DNS traffic are used to identify malicious hosts that are attempting to exploit a particular vulnerability. Furthermore, botnet detection ap-

proaches can be categorized depending on where they are installed and configured into two categories as follows:

- **Host-based Intrusion Detection Systems (HIDS)** include incoming intrusion detection and outgoing intrusion detection. Most botnets scan the exploits, and infect the victims by exploiting particular vulnerabilities of hosts. Knowledge of the signatures and behaviors of the botnets (particularly in infection phase) are used as rules for detection. The main disadvantage of this type technique is being ineffectiveness for the zero-day exploits. Outgoing intrusion detection techniques checks the stealthy malicious outgoing extrusions from a host which are usually unknown by the compromised victims [15]. The challenge of separation between normal behavior and intrusion becomes important in this technique.

- **Network-based Intrusion Detection Systems (NIDS)** aim to identify attacks or malicious behavior by observing network traffic, preferably in real time. The lifecycle of botnets requires maintaining communication between the rallying hosts such as C&C heartbeat traffic, propagation of botnets, and their malicious activities. Depending on the communication protocol of the botnet, network-based approaches such as signature filtering (*e.g.*, host, content-based, protocol, and port) or traffic patterns (*e.g.*, flow level traffic and DNS query analysis) can be used for botnet detection.

Since we will concentrate on network-based detection in our experiments (*cf.*, Chapter 5), we limit the scope of discussion network-based botnet detection methods in this chapter. In general, the principal network-based methods of detecting botnets either rely on passive DNS analysis (*e.g.*, [11] targeting fast flux activity) used to hide the identities of botmaster servers, or rely on deep packet inspection targeting unencrypted malware in-transit (the latter often part of a more comprehensive NIDS system, *e.g.*, [23]). Intercepted malware may be quickly inspected to infer behavioral signatures of the botnet. Netflow information has also been exploited to detect botnets (*e.g.*, [24, 22, 35]). Such analysis can lead to rankings or a reputation system for Internet domains (or individual IP addresses). Recently, analysts have also proposed jointly using netflow and DNS analytical methods *e.g.*, [53].

In the following subsections, with the network-based detection approaches presented by the research community [21, 50, 67, 66], we survey the recent research efforts of network-based botnet defense strategies, and split botnet detection techniques that are identified by the type of data they used for analysis into 3 categories: packet-based ap-

proaches, DNS-based approaches, and flow-based approaches. This chapter gives background information about botnet detection techniques that are useful for understanding of our flow generator framework implementation that we present in Chapter 4, experiments described in Chapter 5, and experimental results given in Chapter 7 .

3.1 Packet-based Detection

Packet-based methods analyze the captured packet data to detect the botnets. In this type of analysis, the payload of the packets, packet size time series, and protocol header information (such as protocol types, source and destination IP addresses, and port numbers) are used for detection. Payload-based detection has many disadvantages including port-number spoofing, privacy issues and payload encryption. In addition, it has limitations in processing high-speed (multigigabit) networks [54]. This type of detection is commonly used in signature-based intrusion detection systems.

Goebel *et al.* [20] implement a passive signature-based IDS technique by analyzing the similarities of the nicknames used by the bots. The TCP packets carrying IRC-related information are captured, and nicknames, IP addresses, port numbers and channel names are stored. Then, a threshold-based score function is implemented based on several definitions such as checking nicknames against defined suspicious substrings, regular expressions, and then n-gram analysis is applied. Based on the result of scoring function, the nickname is added to the white list or black list, and an alarm is generated if necessary. This technique mainly suffers from relying on the payload information for nickname identification, which becomes useless, when payload is encrypted. On the other hand, the use of dictionary words as a nickname will not be detected by the system even for unencrypted traffic.

Kondo *et al.* [34] train support vector machine (SVM) classifier to detect botnet C&C traffic. The feature set does not include packets and protocol header information such as protocol types, port numbers or transport protocol flags; rather they use vectors of session information, packet sequence and packet histogram. The session information vector contains session time, send/receive packet counts and send/receive data size; the packet sequence includes a total of packet size and packet interval time of 16 packets; and the packet histogram vector includes the packet payload size and packet interval time of the send/receive packets. The authors also compare the results with Naive Bayes and k-Nearest Neighbor (k-NN) algorithms; overall they show that SVM gives the best detection results.

Gu *et al.* implement two different payload-based botnet anomaly detection techniques, Bothunter [23] and Botsniffer [24] respectively. The main goal of the BotHunter system is correlating the inbound intrusion alarms with the outbound communication patters to detect the infected host by monitoring the bidirectional traffic between internal hosts and the Internet. For that reason, the system performs a dialog process on five events: inbound port scan, receipt of an inbound exploit, internal to external bot binary download, C&C inbound communication, and C&C outbound port scan. Snort signatures, statistical sCan anomaly detection engine (SCADE) and statistical payload anomaly detection engine (SLADE) are used to detect the five events. The BotHunter system recognizes the infection and coordination dialog that occurs during a successful bot infection. BotSniffer improves BotHunter by attempting to overcome the analysis of encrypted traffic. In addition, it does not use signatures or the identity of the C&C traffic but instead detects the spatial-temporal correlation to identify centralized botnet C&C channels in a monitored network.

3.2 DNS-based Detection

As described in Section 2, DNS queries of C&C servers can be used for botnet detection. Infected hosts access to the C&C server with using its domain name by performing DNS queries during rallying, C&C link failures, C&C server migration, C&C server IP address changes, and during performance of malicious behaviors. By examining the DNS queries or responses, C&C servers and bots can be identified. On the other hand, with the dramatically increasing threat of fast-flux networks, DNS query/response packets are also analyzed to detect either hosts behaving as a reverse proxy or to identify the motherships.

Choi *et al.* [13] propose a botnet detection technique that monitors DNS traffic, which form a "group activity" in DNS queries simultaneously sent by distributed bots. As an example, when a C&C server is updated to a new domain name, it affects all participating bot, and DNS queries. The botnet detection algorithm consists of 3 different parts. Insert-DNS-query stores the features of DNS queries such as source IP, domain name of the query and timestamp of the query. After grouping the names in the dataset by timestamp and domain name, the Delete-DNS-Query component removes the queries such that either the size of IP list do not exceed the size of the given threshold or the domain name is legitimate (*i.e.*, already exists in a whitelist). Detect-BotDNS-Query compares the IP lists of different domain names and generates a similarity score. If after

a period of time a domain has a similarity close to 1, then it is identified as a malicious domain name with the IP addresses stored.

Solomon *et al.* [60] presented two approaches to identify the C&C servers based on anomalous DDNS traffic. The first approach looks for the domain names whose query rates are high. The second approach looks for the abnormal recurring of NXDOMAIN (*i.e.*, an error message indicating that domain is either not registered or invalid) response. NX-DOMAIN-response is sent to the DNS server when domain name cannot be resolved. This situation can occurr when C&C servers have been taken down or during C&C migration. The algorithms of Chebyshev's inequality and Mahalanobis distance are applied to the extracted feature set from DNS responses with specific TTL values (*e.g.*, at most 60, 300 or 600 seconds) either in the A records or in the NXDOMAIN response. These features are then used to identify the C&C servers.

Holz *et al.* [26] propose Flux-Score to detect the domain names whether they are benign or fast-flux hosts. The system is based on the observation of 3 features: unique A records, number of nameserver records, and number of unique autonomous systems (ASNs) for all A records from the DNS queries. Flux-Score detects the benign and fast-flux domains by using a threshold-based decision function. A similar approach, Fluxor, is proposed by Passerini *et al.* [48]. Fluxor system aims to improve the false-positive rates of Flux-Score by applying 9 features instead of 3 features. The Fluxor system consists 3 components: Collector, Monitor and Detector. The Collector harvests domain names from various sources (*e.g.*, spam emails and DNS queries), and marks it as a suspicious domain. The Monitor collects and characterizes the features of suspicious domains, and it enumerates the IP addresses of the agents serving the network for malicious domains. The Detector classifies domain names as malicious or benign by using a Naive Bayes classifier with the given features.

Bilge *et al.* [11] introduce the EXPOSURE system, which detects the domains that are involved in malicious activity by analyzing DNS traffic. The system consists of 4 components: The data collector records the DNS traffic, and feature attribution component extracts the features. A total of 15 features are extracted from the DNS response messages returned from the authoritative DNS server to the recursive DNS (RDNS) servers. The feature set includes time-based, DNS answer-based, TTL value-based, and domain name-based features, and 6 of these features commonly used in detecting malicious Fast-Flux services or in classifying malicious URL. The malicious and benign domain collector collects the domains that are known to be benign or malicious from sources such as blacklists. Then, after the entire dataset is labeled, it is passed to the

learning component which trains the labeled dataset, and when a new domain arrives, it classifies as a malicious or benign by using the extracted features. Thus, the system uses a passive approach which is capable of detecting unknown malware domains that are not in blacklists; in addition, the system might be used to detect different kinds of malicious domains such as phishing sites, spamming domains, and botnet C&C servers which makes EXPOSURE more comprehensive than Fluxor [48]. However, attacker who is aware of the EXPOSURE system may try to change the distribution of the features to evade the detection.

3.3 Flow-based Detection

Flow-based intrusion detection (network behavior analysis) systems examine the traffic statistics of packet flows to identify attacks and malicious behavior in the network [52]. In such flow-based network intrusion detection systems, network flows are identified as a time-proximal sequence of packets with common flow keys: source IP address, destination IP address, source port number, destination port number, and protocol (*e.g.*, IP 5tuple). While flow-based intrusion detection systems provide an aggregated view of network traffic features and reduce the amount of data that need to be processed, traditional intrusion detection systems are designed to inspect the contents of every packet for known signatures of attacks (*e.g.*, packet-based detection, particularly inspecting payloads). In addition, payload-based IDS has limitations including the violation of privacy laws, inability to process simple encrypted payload information, and demands very significant computation and memory bandwidth of hardware in high-speed (multigigabit) networks. Therefore, flow-based methods are accepted as a promising and complementary approach to conventionally used payload-based inspection methods for botnet detection [54].

Livadas *et al.* [38] apply machine learning algorithms by extracting features from network flows to identify the IRC botnet C&C traffic. The authors use two stage approaches in their experiments. In first stage, they classify the communication flows as either IRC or non-IRC traffic. This stage aims to reduce the number of flows that can represent the botnet flows. In second stage, the separation of botnet and legitimate chat flows are identified. The authors perform experiments using C4.5, Naive Bayes and Bayesian Network classifiers by applying the dataset extracted and labeled from Dartmouth campus and simulated Kaiten botnet. The feature set is extracted from the communication traffic, and includes various statistics of flows ranging from timing-based features such as duration, and flow start end times to packet header statistics such as

maximum initial congestion window size, variance of bytes per packet in a flow.

Strayer *et al.* [57, 56] perform IRC-based botnet detection by examining the flow characteristics in 4 stages. In the filtering stage, they filter the traffic that is unlikely to be part of the botnet traffic. In order to reduce the number of flows, the filtering stage consists various filters. First, the TCP flows with a successful 3-way handshake are kept, and all other protocols are discarded. Second, they discard the port scanning traffic and high-bit-rate flows such as bulk data transfer or P2P traffic. And as a final filter, the packets that have greater than average size of 300 bytes in a flow, and the flows less than 2 packets or less than 60 seconds duration are removed. This stage dramatically eliminates the number of flows that are passed to the classification stage. Similar to [38], the authors apply machine learning algorithms to the extracted flow feature dataset. In correlation and topology stages, a pairwise comparison of the flows is performed to analyze similarities of the flows in order to identify the zombies in the same botnet and the identity of the bot controller (*i.e.*, IRC server).

Karasaridis *et al.* [32] propose an anomaly-based passive analysis algorithm to detect IRC C&C center by using flow statistics. The strength of the proposed solution is detecting IRC botnet controllers that are running random ports without any signatures and payload information. In addition, it is invisible to operators, and can be used in encrypted traffic. The authors isolate flows of the suspicious hosts, then the flow records stored as a summary of flow record between a local and remote host (candidate controller conversations (CCC)). To identify the suspicious hosts, the authors use the well-known IRC ports, a flow model for IRC traffic that represent typical C&C activity, and an identification approach by using the hosts that have multiple connections from many suspected bots to one or more of their local ports. Each of the CCC consists of flow features and supplementary data such as number of packets, number of bytes, remote port number and timestamp of the first and last flow for further analysis. After constructing the CCC flows, the authors analyze the record to isolate the suspected hosts and controllers. The CCC analysis consists of 3 different parts. First part is calculation of the unique suspected bots for a given remote server and port. Second part is calculating the Euclidean distance of the model traffic and observed traffic by using 4 metrics extracted from the flows such as flows-per-address, and packets-per-flow. In final part, heuristic scores for server address and port pairs are calculated to detect suspicious candidates by extracting the number of idle clients that have certain pattern and whether server uses both TCP and UDP on suspected port or server appears to be serving P2P traffic. Finally, based on the analysis alarms and report are generated.

Mesud *et al.* [41] propose a multiple log-file based temporal correlation technique for detecting C&C traffic. The dataset consists of flow features generated from the tcpdump log files and correlation of packet features with the exedump logs which includes start times of the applications. The authors create a correlation between the packets and the exedump log files. As an example, by checking the response time of the incoming packets at clients, different heuristics such as bot response with a Boolean value are applied. After enriching the dataset in addition to flow features, different machine learning algorithms are applied to test their approach on two different IRC-based bots. The algorithm works for only IRC-based botnets and it is easy to evade the correlation of packets and timing-based features.

Gu *et al.* [22] present a general botnet detection method (BotMiner) that is independent of the botnet C&C protocol (*e.g.*, IRC-based, HTTP or P2P). The BotMiner system monitors both C&C communication (C-plane) and malicious activities (A-plane) for botnet detection. The C-plane monitors the network traffic, and captures the TCP and UDP flow records based on who is talking to whom. The flow features include the information related to flow time, flow duration, number of packets, source, destination IP addresses and port numbers (C-flows). The A-plane monitors the logs information about the hosts by analyzing malicious activities of the outbound traffic. C-plane clustering performs the filtering and clustering by reading the log generated by the C-plane monitor. First, C-plane clustering discards the flows that are not directly from internal hosts to external hosts, contains one-way traffic, and whose server IP record is a legitimate server. Then, it extracts 4 different statistical flow features from each C-flow as follows: the number of packets per hour, the number packets per flow, the average number of bytes per packet and the average number of bytes per second. Finally, the flow vectors are clustered by using X-means clustering algorithm in order to group the hosts that have similar C&C communication patterns. A-plane clustering performs clustering on activity logs generated by Snort [51]. The hosts are clustered first according to the types of their malicious activities (*e.g.*, scan, spam, and binary downloading). Then each host is clustered for a specific activity features. After obtaining the cluster results from both C-plane and A-plane, the BotMiner system then performs cross-plane correlation to merge the results and identify the hosts with common communication and malicious activity patterns. The BotMiner system is tested with 8 different botnet traces that are captured from the wild and simulated in a controlled environment. The system performs with a very low false positive rate with different types of botnets (*i.e.*, HTTP, P2P and IRC).

Chapter 4

Flow Generator Framework

A network flow is defined as "a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties" [49]. To identify a particular TCP or UDP flow, the common properties from the packets' layer 3 and 4 headers can be defined as 5 tuples: source IP address, source port, destination IP address, destination port, and transport layer protocol.

Extracting the flows heavily rely on the ability of network devices to generate and export the flow information. NetFlow (developed by Cisco Systems) [2] is a commonly used standard for IP flow monitoring and export. Other major vendors also built their own flow-enabled devices. As an example sFlow is used by 3Com, D-Link, Hitachi, Hewlett-Packard and NEC, Jflow is used by Juniper Networks, and Huawei Technology uses NetStream. The Internet Engineering Task Force (IETF) standardized NetFlow 9 under the name of IPFIX (IP Flow Information Export)[14]. A typical flow system consists of four components: *flow exporter*, *flow collector*, *flow reporter*, and *analyzer*. Flow exporter captures statistics from the packet headers that are observed from the routers and switches in the network. Then, it transmits captured statistics in the form of User Datagram Protocol (UDP) or Stream Control Transmission Protocol (SCTP) to the flow collector, and then the statistics are further processed for analysis and archiving.

Flows can be identified as a particular instance of communication between source and destination hosts by using 5-tuples that are proximal in time. Each flow is stored in a vector of flows instead of storing the packet traces for deep packet inspection (DPI). The flow vectors are well suited to represent the interactions between host applications in a network (based on the specific set of statistical features of flows (*e.g.*, average packet

size and flow duration)). Flow-based vectors (*i.e.*, constituting dataset) may be applied to visualize traffic patterns associated with individual routers and switches as well as on a network-wide basis (traffic aggregation, application based classification) to provide proactive intrusion detection, efficient troubleshooting, and rapid network problem resolution [5]. As an example, flow information may be applied to detect the presence of worms, port scans, DDoS attacks, botnets, and other security threats by applying clustering, or classification to the exported flow statistics. Furthermore, different reflection of server port numbers and flow features can be used to detect anomalies that may indicate malicious activity.

In the following subsections, we present a TCP-based flow tool which generates flow-based dataset from the traces in a PCAP format. The dataset consists of a total of 21 features and can be used for network security analysis. We begin by presenting the components of the flow generator framework to illustrate the flow generation process. After exploring the design of the framework, we then present the extracted features and log event reports.

4.1 Design

4.1.1 Framework of Flow Generator

There are many open source projects (*e.g.*, argus[1], netdude[2], flowstat[3]) that generate the flows from packet traces or process the native NetFlow data. However, these tools are restricted to extract on demand flow features such as push bit count or number of RTT samples. On the other hand, network traffic generators such as Tmix[63] and Swing[61] can be used to obtain the flow features. However their design is specific to generate realistic traffics, modifying and adapting new functions to these systems adds new challenges. For these reasons, we present a flow generator framework to generate flows from raw PCAP files which is a more efficient and convenient way to extract features by defining rules and filters for our experiments.

Figure 4.1 shows the architecture of our flow generator tool which consists of 3 main components: user interface, stream generator, and flow generator. Algorithm 1 summarizes the entire work flow of feature generation process. The constraints considered for defining flow generation configurations were correlated to determination of flows to be exported to the collector for feature extraction and analysis. Flow generator tool is

[1]http://www.qosient.com/argus/
[2]http://netdude.sourceforge.net/
[3]http://www.splintered.net/sw/flow-tools/docs/flow-stat.html

implemented using bash scripting, perl and C++. Since the flow generator tool uses the *libpcap* packet sniffing library by utilizing tshark[4], new filtering mechanism (suh as botnet and background traffic RTT extraction are used to estimate RTTs in Section 5.2.5), and supplementary modules can be added easily.

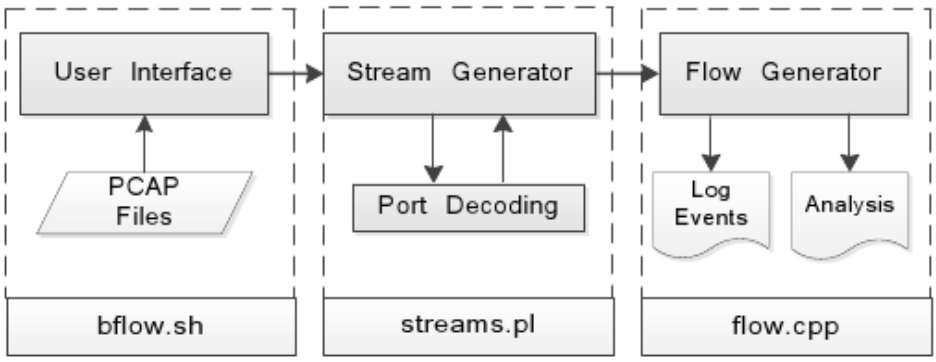

Figure 4.1. Flow Generator Components

As shown in Figure 4.2, user interface, *bflow.sh*, takes flow metrics (protocol type, number of packets and flow timeout) as an input and automatically starts processing the PCAP files under the same directory. User interface communicates with two components for flow generation and feature extraction. Stream generator, *streams.pl*, automates the stream generation by using the tshark libraries through using the system command. It is responsible for analyzing bi-directional session between two endpoints depending on their protocol type, and then it automatically collects the set of packets between the two endpoints which have a set of common properties: source IP, destination IP, source port, destination port and protocol (IP 5-tuple). Then, stream generator sends the streams to the flow generator when one of the following conditions has occured within given user metrics or as a design default:

- **Flow Inactivity Timeout:** The inactivity timer measures the length of time expired since the virtual router recorded the last datagram for a given flow. When this timer expires, the virtual router exports the flow cache entry from the cache and removes it. When, at a later time, another datagram begins that uses the same flow characteristics, the virtual router allocates a new flow cache entry, and the inactivity timer begins again. The possible range for the inactivity timer is from 10 to 600 seconds. For example for Cisco Netflow the default value is 15 seconds [2].

[4]http://man-wiki.net/index.php/1:tshark

```
Usage: sh bflow.sh [-hv] - p [tcp] -n [number] -t [timeout]
This program reads PCAP files in a current directory and generates the flow features for further analysis

OPTIONS:
        -h              Help
        -p <tcp|udp>    Transport protocol type
        -n <num>        Number of packets, first n packets after 3 way handshake
        -t <secs>       Flow inactivity timeout
        -v              Version
```

Figure 4.2. User Interface

- **Number of Packets:** First n packets will be used to export the flows to the flow generator. For TCP protocol depending on the use of flow features, n can be adjusted to the first n packets after or including 3 way handshake. Partial flows are processed through the first observed packet.
- **FIN or RST Flag:** FIN or RST TCP flags indicate that flow is terminated, and it is ready to export to the flow generator.
- **Flow-cache Memory:** When the flow-cache memory gets full, algorithms such as least recently used (LRU) or heuristic algorithms, depending on port numbers and protocols, are used to export the flows to the flow generator. In our design, we do not consider this option, and solve the problem by splitting the given PCAP files into smaller sizes that consists less packets.

Algorithm 1 Flow Generation Algorithm

Input: user metrics, traces in .PCAP format
Output: flow.out and log.out files

```
 1: if UserMetrics then
 2:     Extract packet exchange between each unique 5-tuple as a source and destination
 3:     Decode port numbers
 4:     repeat
 5:         if FlowSuccessful then
 6:             Dissect TCP packet fields
 7:             Remove from cache, and call Feature Generator
 8:         else
 9:             Terminate flow and save reports to log.out file
10:         end if
11:         if FlowFeatureTrigger then
12:             Calculate flow features
13:             Save flow features to flow.out file
14:         end if
15:     until AllFlowsProcessed
16: end if
```

Flow generator is responsible for aggregation of statistics from the streams, and storing the flows as a vector for further experiments. It extracts the features from the streams, and saves features and log events to *flow.out* and *log.out* files respectively. From given PCAP file, we construct the flow vector, $V_F =< F_1, \ldots, F_n >$, and save 14 features to the *flow.out* file in the following form:

$$F = (Duration, IP_{src}, IP_{dst}, port_{src}, port_{dst}, f_1, \ldots, f_n, label_{src}, label_{dst})$$

where the flows are defined with source and destination IP numbers (IP_{src}, IP_{dst}), port numbers ($port_{src}, port_{dst}$) and protocol. $f_1, \ldots, f_n$, represents the static and dynamic flow features (discussed in Section 4.1.2), and *duration* of a flow defined the time difference between the last packet and first packet in seconds (*i.e.*, the observation time of the first SYN set packet from source to initiate the TCP connection). We will consider only successfully established TCP flows whose three-way-handshake were successful. In our default settings the system analyzes the TCP packets after 3-way handshake; we identify the source and destination by checking the initial SYN or the SYN+ACK flags of a TCP connection as well as well-known port numbers of each flow by checking the port number of the destination. We specify the application type by extracting the client and server port numbers for each flow, we then apply the Internet Assigned Numbers Authority (IANA)'s list of registered ports [3] to determine the application types. Log events, which gives suplementary information about the unsuccessful flows are, saved to the *log.out* file in the following form:

$$L = (T_{start}, IP_{src}, IP_{dst}, Port_{src}, Port_{dst}, label_{src}, label_{dst}, status)$$

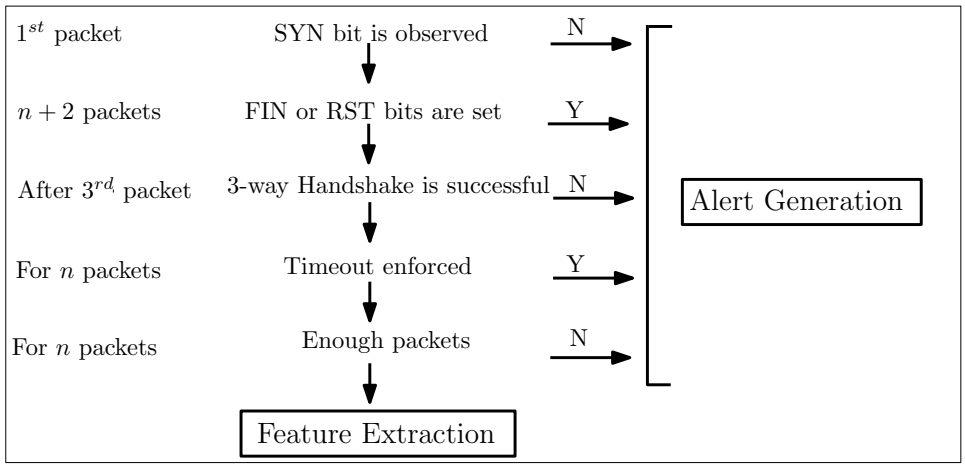

Figure 4.3. Alert Generation

log.out file is used for alert generation, it reports the status alerts by checking all constraints in a sequential order as shown in Figure 4.3. If flow feature extraction is not successful, flow generator saves the log events with a status defined as follows:

- **Partial Flow:** It shows that there is no 3-way handshake is observed between source and destination.
- **FIN or RST Set:** It defines that the flow is terminated if only FIN or RST bits are set during the time that flows are processing, and if one of these flags are set in final packet (n + 3-way handshake), the flow is imported successfully to flow.out file.
- **3-way Handshake not Complete:** In some cases even if SYN bit is seen from the source, the 3-way handshake is not complete, as an example we observed SIP (Session Initiation Protocol) protocol in which source sends more than one SYN bit set packets to destination, this filter eliminates these type of conversations.
- **Timeout Enforced:** If time between consecutive packets exceeds some inter-flow gap, the flow is terminated and reported in log.out file.
- **Not Enough Number of Packets:** If two endpoints have less than $n+3$ packets, flow is terminated and reported in log.out file.

4.1.2 Flow Level Features

In [43], the authors describe 248 flow-level features. Among these features, we focus on 10 features: cnt-data-pkt, min-data-size, avg-data-size, init-win-bytes, RTT-samples, IP-bytes-med and frame-bytes-var, pushed-data-pkts (see Table 4.1 for definitions). According to [37], these features were selected from among 248 features after applying a correlation-based filtering mechanism on each of the datasets. We also identified the two "derived" features, *IP-ratio* and *goodput*, as promising ones not strongly dependent on TCP. We enrich these set of features with total connection time (*i.e.*, duration), IP addresses and port numbers (both undecoded and decoded).

The extracted features can be defined in two categories: static features and dynamic features. Flow duration is an example of static feature that is not carried in the packet header over the lifetime of the flow, while dynamic features are likely to change as the flow progress through time. Dynamic features can be calculated from both using the packet headers and payload information. Furthermore, other features can be drawn from using both dynamic and static features. As an example, goodput feature is calculated

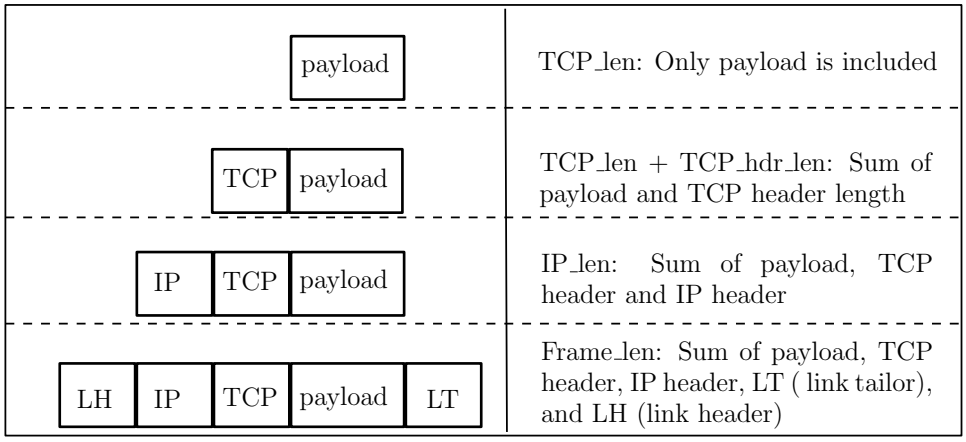

Figure 4.4. Terms Used in Feature Extraction

by dividing the total number of frame bytes by flow duration. In our flow generator tool, we extract all features from packet headers, and also enrich these features with supplementary information such as unique flow keys (*e.g.*, IP addresses, port numbers). While calculating the flow features, to make the terms related to packet encapsulation clear, we illustrate the terms that are used in flow generation system in Figure 4.4.

Nr.	Abbreviation	Description	Properties
1	Duration	Total connection time	Time difference between the last packet and first packet (SYN flag is seen from destination)
2	IP-src	Source IP address	-
3	IP-dst	Destination IP address	-
4	src-port	Source port number	Undecoded
5	dst-port	Destination port number	Undecoded
6	cnt-data-pkt	The count of all the packets with at least a byte of TCP data payload	-TCP_len is observed -Client to server
7	pushed-data-pkts	The count of all the packets seen with the PUSH set in TCP header	-Client to server & server to client
8	min-data-size	The minimum payload size observed	-TCP_len observed -Client to server -0 if there are no packets

Continued on next page

Table 4.1 – continued from previous page

Nr.	Abbreviation	Description	Properties
9	avg-data-size	Data bytes divided by the total number of packets	-TCP_len observed -Packets with payload observed -Server to client -0 if there are no packets
10	init-win-bytes	The total number of bytes sent in initial window[5]	-Retransmitted packets not counted -Client to server & server to client -0, if no ACK observed -Frame_len calculated
11	RTT-samples	The total number of RTT samples[6] found	-Client to server
12	IP-bytes-median	Median of total IP packets	-IP_len calcualted -Client to server
13	frame-bytes-var	Variance of bytes in Ethernet packets	-Frame_len calculated -Client to server
14	IP-ratio	Ratio between the maximum packet size and minimum packet size	-IP_len calculated -Client to server & server to client -If one packet observed 1, and if no packets are observed 0 will be reported
15	goodput	Total number of frame bytes divided by the differences between last packet time and first packet time	-Frame_len calculated -Client to server -Retransmitted bytes not counted
16	Srclabel	Source port label	-Decoded port number
17	Dstlabel	Destination port label	-Decoded port number

Table 4.1: Flow Feature Definitions and Descriptions

[5]Initial window defines the number of bytes seen before the first ACK packet from the destination[43].

[6]An RTT sample is found only if an ack packet is received from the other endpoint for a previously transmitted packet such that the acknowledgment value is 1 greater than the last sequence number of the packet. Further, it is required that the packet being acknowledged was not retransmitted, and that no packets that came before it in the sequence space were retransmitted after the packet was transmitted [43].

Chapter 5

Problem Statement and Experiment Set-up

5.1 Problem Statement

As detailed in Chapter 3.3, flow-based classification and clustering methods examine traffic statistics of packet flows to identify attacks and malicious behavior in the network. Among these methods, botnet detection techniques extensively perform ML algorithms by using timing-based flow statistics. As shown in Table 5.1, [38], [57] and [56] extract duration, average packet rate, and bit rate per second characteristics of flows for performing supervised classification. A recent approach [22] similarly uses the number of flows per hour and the average number of bytes per second for performing unsupervised clustering.

These techniques use representative attack traffic for both training and testing their classifiers to provide confidence in their proposed solutions. However, there are not many realistic datasets publicly available with background and attack traffic naturally present together. For these reasons, researchers often generate synthetic malicious traffic to salt real enterprise packet traces for evaluating the effectiveness of the proposed classifiers. As an example, [38] uses a testbed to obtain the botnet traces, and [56, 57] use more representative botnet traces by hosting bots inside the internal network and a C&C center on an external network. The BotMiner system [22] salts two IRC and two HTTP botnet traces, which were captured by running benign versions of bots in a simulated environment to the background traffic.

We explore how the timing-based features determined during the salting process

System	Timing-based features	Botnet Traces	Detection
Livadas*et al.*[38]	-flow start/end times -flow duration -average bits per second for flow -average packets per second for flow	Simulated IRC-based botnet Kaiten	-C4.5 -Naive Bayes -Bayesian Network
Strayer *et al.* [57, 56]	-flow start/end times -flow duration -average bits per second for flow -average packets per second for flow	Simulated IRC-based botnet Kaiten (C&C center resides external network, and nine bots are within internal network)	-C4.5 -Naive Bayes -Bayesian Network
Karasaridis *et al.* [32]	-timestamp of first and last flows of the conversation	Not specified	-Anomaly-based detection
Mesud *et al.* [41]	-incoming and outgoing packet timestamp of monitored host (applying to each packet with a threshold time value to aggregate the packets)	Simulated IRC-based botnet Kaiten	-Support vector machine (SVM) -Bayesian Network -C4.5 -Naive bayes -Boosted decision tree (Boosted J48)
Gu *et al.* [22]	-number of flows per hour -average number of bytes per second	Simulated two IRC-based (V-spybot and V-sdbot), two HTTP-based (implemented by analysis), and 4 real world (IRC, and P2P) botnets	-Clustering

Table 5.1. Recently Proposed Flow-based Botnet Detection Approaches

may affect classification performance. Timing-based features may vary depending on the position of the packet-trace monitor and, of course, on the geographical separation of communicating points including bots and their masters (and intermediate stepping stones if any). In both situations, "directly salting botnet traces into the background traffic may introduce timing inconsistencies with the background traffic which may affect classification performance. Hence, it is necessary to achieve timing consistency between the traces to mitigate specific artifacts of timing-based features. For these reasons, we analyze the factors that contribute to the timing-based features of flow classifiers. After simulating a benign version of the Kaiten bot to acquire botnet C&C traces, we estimate the eligible RTT samples from the background traffic. We calibrate the simulated botnet traffic timestamps to provide consistency of timing-based features between the traces. Then, background traffic is salted to the simulated botnet traces, and then flow features are extracted.

As shown in Figure 5.1, Chapter 5 details the experiment set up. We begin by

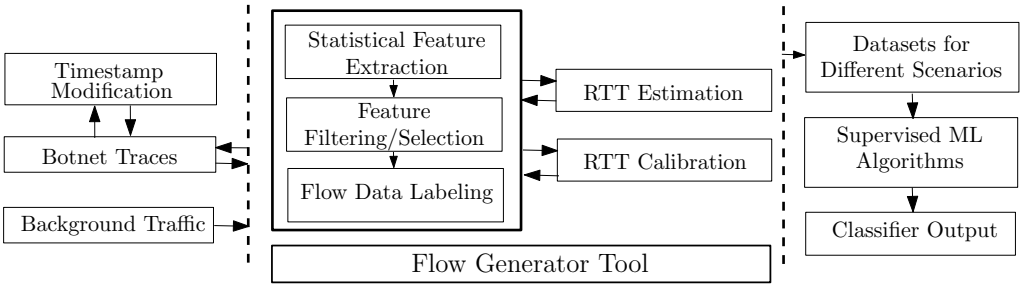

Figure 5.1. Workflow of Experimental Set up and Classification

describing the LBNL public packet trace, flow generation procedure, flow features we extracted, the way ground-truth flow-class labels were derived, how the Kaiten bot was simulated, RTT calibration and botnet trace timestamp modification. Chapter 6 provides a description of ML algorithms that are used in our experiments. We then describe the datasets of four different scenarios, and we evaluate how sensitive the supervised ML algorithms to timing-based features by analyzing the classification results of each algorithms in Chapter 7.

5.2 Experiment Setup

5.2.1 Packet Trace Data

In our experiments, we use publicly available Lawrence Berkeley National Laboratory (LBNL or now just LBL) packet traces [4]. According to [46], the LBNL trace represents internal enterprise traffic recorded at a medium-sized site. The measurement system at LBNL simultaneously monitored the traffic out of two of the (more than twenty) router ports and collected packet traces that spanned more than 100 hours of activity from a total of several thousand internal hosts. Thus, the packet traces are from a successive sampling of subnets. All together, 11GB of packet header traces from October 2004 through January 2005 are available for analysis. This data was publicly released and anonymized by the system described in [47, 46]. The authors used a prefix-preserving scheme to remap the external and internal IP addresses. Additionally, the subnet and host portions of the internal addresses were further transformed in a one-to-one fashion. However, the port numbers in the TCP headers are intact, which provides information about application type at the time the trace was recorded. The authors also provided meta-data allowing us to obtain anonymized subnets and gateway addresses which are used for RTT estimation in our experiments.

The raw trace used in our experiments is a combination of those recorded on Oct. 4, 2004, router port 19, and on Dec. 15, 2004, router port 8[1].

5.2.2 Flow Generation and Flow-level Features

By using the flow exporter system explained in Chapter 4.1.1, we define the flow metrics (protocol type, number of packets and flow timeout) as an input and automatically process the LBNL PCAP files. First, the system analyzes bi-directional session between endpoints depending on their protocol type, then it automatically collects the set of packets between the two endpoints which have a set of common properties: Source IP, destination IP, source port, destination port and protocol (IP 5-tuple) [49]. Since the simulated Kaiten Bot uses TCP-based attacks, we restrict our attention to TCP flows. Following [37], we extracted features from first 5 packets after the 3 way handshake with 60 seconds inactive timeout value[2] to prevent significant processing load of capturing and exporting per flow information at the monitoring device. In addition, the first 5 packets provide feasibility of detection in early stage of a connection [10].

In [43], the authors describe 248 flow-level features. Among these features, we focus on the eight features: cnt-data-pkt, min-data-size, avg-data-size, init-win-bytes, RTT-samples, IP-bytes-med and frame-bytes-var (see Table 5.2 for definitions). According to [37], these features were selected from among 248 features after applying a correlation-based filtering mechanism on each of the datasets. We also identified the following two "derived" features as promising ones not strongly dependent on TCP:

- **IP-ratio:** The ratio of the maximum to the minimum IP packet size (client to server and server to client),
- **Goodput:** Total non-duplicate number of frame bytes divided by total flow time (client to server and server to client).

Using these twelve features, we extracted a total 7266 LBNL TCP flows. Note that the twelve features we extract do not consider fields that can be readily tampered with, such as *port numbers* or protocol-specific information in the payload header, *e.g.*, count of packets with the push bit set in the TCP option field [38]. Therefore, a classifier built

[1] From [4], files lbl-internal.20041004-1458.port019.dump.anon and lbl-internal.20041215-0510.port008.dump.anon.

[2] The inactivity timer measures the length of time expired since the monitor recorded the last datagram for a given flow. When this threshold expires, the monitor exports the flow. As an example, the default value for NetFlow [2] is 15 seconds.

Abbreviation	Description
cnt-data-pkt	The count of all the packets with at least a byte of TCP data payload (client to server)
min-data-size	The minimum payload size observed (client to server)
avg-data-size	Data bytes divided by number of packets (server to client)
init-win-bytes	The total number of bytes sent in initialwindow (server to client client to server), see [43]
RTT-samples	The total number of round-trip time (RTT) samples found (client to server), see [43]
IP-bytes-med	Median of total bytes in IP packet (client to server)
frame-bytes-var	Variance of bytes in (Ethernet) packet (server to client)
IP-ratio	Ratio between the maximum packet size and minimum packet size (server to client & client to server)
goodput	Total number of frame bytes divided by the flow duration (server to client & client to server)

Table 5.2. Flow Feature Definitions and Descriptions

based on these features should tend to be more "tamper-resistent" than one based on highly TCP-dependent features [68].

5.2.3 Ground Truth Labels

For supervised classifier training, it is necessary to ascertain ground-truth class labels for both training and test data. However, LBNL traces have only the anonymized raw packets publicly available, containing only the TCP/IP header information without any payload. Fortunately, port numbers were unaffected by the anonymization process, with the exception of traffic associated with one particular port used as an internal security monitoring application. We extracted the client and server port numbers for each flow, and then applied the Internet Assigned Numbers Authority (IANA)'s list of registered ports [3] to determine the application. We observed that the LBNL trace includes a large variety application types with different communication patterns. As an example, web, e-mail and name services include both enterprise network and wide-area network traffic. Windows services and network file services are the only within enterprise applications. We labeled every flow in the dataset with the corresponding application category, following the same procedure as [40]. While port-based application identification might seem ineffective, the use of non-standard or dynamic port numbers is unusual enough

in the LBNL enterprise traffic that it does not affect our label definitions significantly [40]. The application category breakdown of TCP traffic is shown in Table 5.3, and the definitions of application classes with sample applications are given in Appendix A.

Class	**Protocols**
bulk	FTP, HPSS
email	SMTP, IMAP4, IMAP/S,POP3, POP/S, LDAP
interactive	SSH, telnet, rlogin, X11
name-srv	DNS, netbios-NS, SrvLoc
net-file	NFS, NCP
net-mgnt	DHCP, ident, NTP, SNMP, NAV-ping, SAP, NetInfo-local
web	HTTP, HTTPS
windows	CIFS/SMB,DCE/RPC, Netbios-SSN, Netbios-DGM
misc	Steltor, MetaSys, LPD, IPP, ORACLE-SQL, MS-SQL

Table 5.3. Flow Class Definitions and Sample Applications

5.2.4 Botnet Simulation

In order to obtain traces of actual botnet C&C traffic, an IRC-based bot Kaiten whose source code was available on the web [1] was simulated. A virtual network of one bot controller, three infected machines (bots) was implemented by reverse engineering the bot code. Figure5.2 shows the topology of the analyzed botnet traces of controlled simulated environment. 3 bots communicate with the C&C server on port 6668, and C&C server port number can be dynamically changed by the botmaster. We collected C&C TCP flows associated with our botnet using wireshark [7] at the simulated IRC server.The complete simulation is described in the technical report [65].

Bots which connect to the IRC server using a predetermined C&C channel, and send a NICK IRC message to convey that the user is online with a certain ID. The controller then replies to verify that the bot is alive and has joined the C&C channel. After the verification phase, the bots wait for commands from the controller. Upon the arrival of a command, the bots act accordingly. If no command is received for 20 minutes, a bot reopens a connection to the controller. Our goal here is to generate C&C heartbeat traffic of an already established botnet that is *readily* standing for commands. Our goal is not intrusion prevention or detecting of bots after an overt attack *e.g.*, denial-of-service (DOS) or email spam has been launched. Instead, we wish to simulate and detect a botnet that is either in a sleep or "stealth" state where, as an example of the latter, the

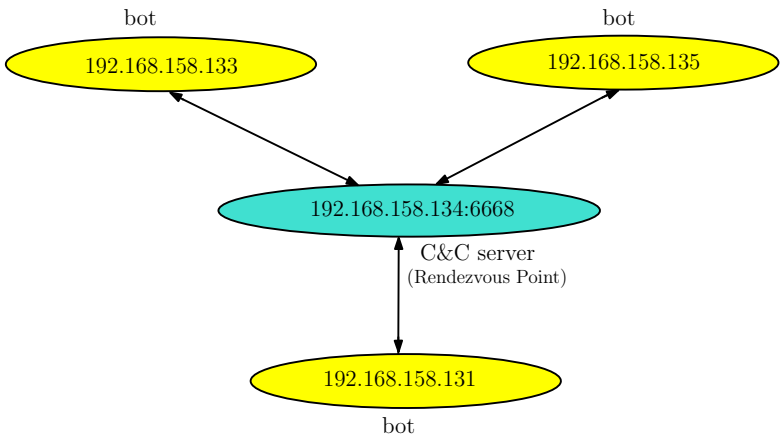

Figure 5.2. Actors in IRC-based botnet Kaiten in Simulated Environment

botnet could be slowly/discreetly exfiltrating private data.

5.2.5 Methodology

In our flow classifier implementation, we used a timing-based goodput feature in both directions (client to server and server to client) defined as the total non-duplicate number of frame bytes divided by the total flow duration. From Figure 5.4, we split bi-directional flows into two unidirectional flows which contain k nonduplicate packet series, $P_{cs} = (P_i, P_{i+1}, \ldots, P_k)$, with a corresponding timestamp, t_i, of the i^{th} packet at the monitoring point. In our experiments, we calculate the goodput feature after the 3 way handshake. Therefore, we do not include RTT samples of the SYN-SYNACK segments and the corresponding SYNACK-ACK segments. We can calculate the goodput feature from client to server as follows (the goodput feature from server to client can be computed in a similar way):

$$Goodput_{cs} = \frac{\sum_{i=1}^{k} frame\ bytes_{cs}}{flow\ duration} \tag{5.1}$$

Equation (5.1) depends on the flow duration which includes the RTT and inter-packet arrival time of the connections. RTT is the difference between the capture time of the sending packet with a certain sequence number (SEQ) and the corresponding follow up acknowledgement (ACK) from the receiver. To avoid the ambiguity of computing the RTT, retransmitted packets are omitted in our goodput calculation. The packet inter-

arrival time is the difference between the time of the i^{th} packet and the $(i-1)^{th}$ packet at the monitoring point. Ideally, it is characterized by the network, application process time and the size of the congestion window. In addition, for specific applications, it includes "thinking" or "reading" time of the users. In this paper, we only deal with the differences of the RTT samples between botnet and background traffic[3]

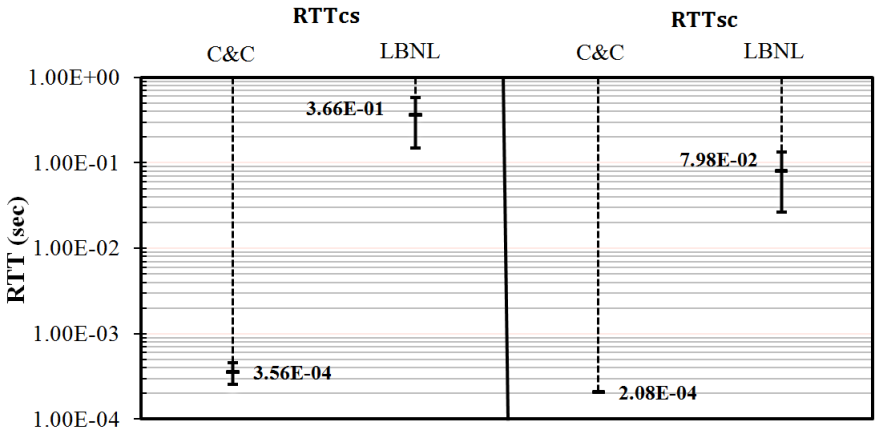

Figure 5.3. Comparison of Background Traffic and Simulated C&C RTTs Before Calibration

Figure 5.3 shows a logarithmic scale plot of mean and standard error of the mean[4] for botnet and background traffic RTT samples before calibrating the simulated botnet C&C traffic. There is a significant difference between the mean values of C&C traffic and background traffic $RTTs$. This explains that goodput features of the botnet traces are likely to change significantly after C&C RTT calibration. RTT samples of the C&C traffic have less within-group difference, since network conditions do not have any effect on the virtual simulated environment used to generate C&C traffic. Additionally, LBNL traffic RTT_{sc} values are considerably smaller than that of RTT_{cs} values, because the position of the LBNL packet-trace monitor is close to the LBNL clients. Now, we are interested in estimating eligible RTTs from background traffic to calibrate the botnet traffic t_i values before salting. There are various methods suggested by the research community to estimate passive RTT values from a packet trace. In [30], the authors proposed SYN-ACK (SA) and Slow-Start (SS) estimation techniques for unidirectional flows. In [59], two different techniques are offered, both of which depend on the TCP timestamp

[3]Packet inter-arrival time statistics are widely used in network intrusion detection systems and Internet application identification. We leave it as a future work to analyze the effects of differences in packet of inter-arrival time on flow classifier accuracy.

[4]Standard error of the mean (SE) is the standard deviation of the sample mean, and defined as $\sigma/\sqrt{N}$ where σ is the standard deviation of samples and N is the sample size.

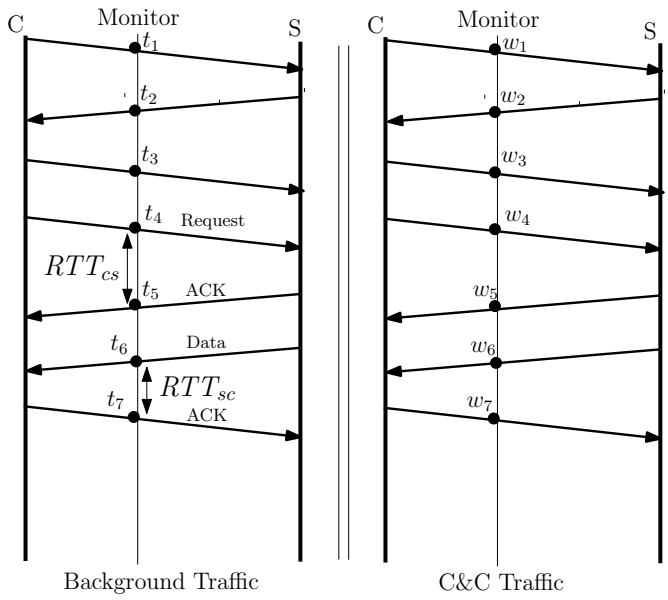

Figure 5.4. Timestamp Modification Procedure

option by adding timestamp value (TSval) and timestamp echo reply (TSecr) values [28] to the TCP header option; however, these methods require a *priori* deployment, and implementation of timestamp increment should be consistent across different hosts. Other techniques [39, 29] generally depend on more complex deduction methods based on the sender's congestion window size, or maximum-likelihood estimation for matching data segment with ACK segment. In this study, we measure the RTT samples using provided timestamp values recorded at the monitoring point. We will use the following procedure to estimate the RTT samples from background traffic:

- We take the IP addresses which reside inside the enterprise as client and the external IP addresses as server. We simulate the external destination IP addresses as a C&C center and IP addresses inside the enterprise as bots.
- We construct RTT vectors of background traffic, $V_{RTT} =< RTT_1, \ldots, RTT_n >$, for both directions corresponding to the eligible subset of the RTTs used in flow feature goodput generation.
- We calculate the empirical cumulative distribution function (CDF) from the RTT samples of the background traffic and use it to randomly generate the RTT_i values for the simulated botnet C&C traces.

After RTT estimation, we calibrate 50 conversations between the bots and the C&C

Algorithm 2 RTT Calibration

1: $k \leftarrow number\ of\ packets$
2: $i \leftarrow 1$
3: **repeat**
4: **if** $(P_i^{sc}\ and\ P_{i+1}^{sc})||(P_i^{cs}\ and\ P_{i+1}^{cs})$ **then**
5: $w_{i+1}^{new} = w_i + (w_{i+1}^{old} - w_i)$
6: **else if** $(ACK^{sc} \in P^{cs})$ **then**
7: Randomly select $RTT_{cs},\ RTT_{cs} \in V_{RTTcs}$
8: $w_{i+1} = w_i + RTT_{cs}$
9: **else if** $(ACK^{cs} \in P^{sc})$ **then**
10: Randomly select $RTT_{sc},\ RTT_{sc} \in V_{RTTsc}$
11: $w_{i+1} = w_i + RTT_{sc}$
12: **end if**
13: $i \leftarrow i + 1$
14: **until** $i = k$
15: $saltToBackgroundTraffic()$

channel traffic by modifying their timestamp values. From Figure 5.4, we divide bi-directional packet flows into two unidirectional flows, P_{cs} and P_{sc}, which contain k nonduplicate packet series of C&C traffic. The initial timestamps of the C&C packets after 3 way handshake, w_3, stays same and the remaining timestamps, $w_4, \ldots, w_k$, of the packets are only calibrated when the ACK of sending packet is received. We calibrate the RTT samples (RTT_{cs} and RTT_{sc}) of the C&C traffic from the corresponding V_{RTTcs} *and* V_{RTTsc} vectors of the LBNL background traffic. As an example the timestamp of the first acknowledgement packet of the C&C traffic, w_5, is calibrated by calculating $w_4 + RTT_{cs}$, where RTT_{cs} is estimated from the LBNL background traffic. The calibration algorithm can be summarized as in Algorithm 2.

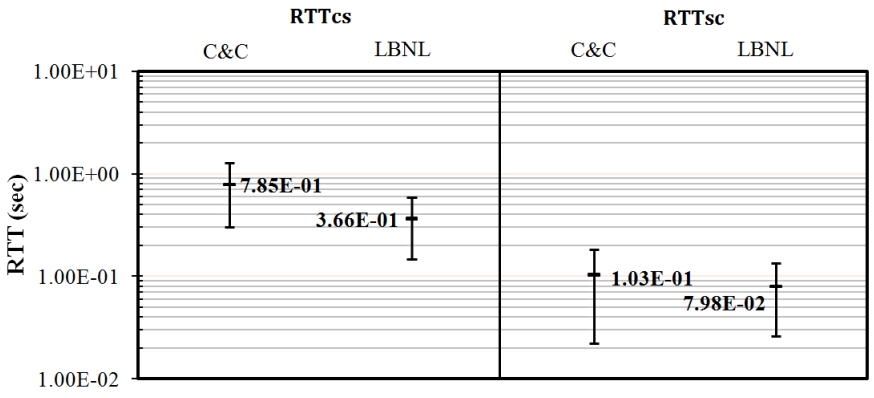

Figure 5.5. Comparison of Background Traffic and Simulated C&C RTTs After Calibration

As a result of application of Algorithm 2 to the RTTs of the botnet C&C traffic, there is better agreement between RTTs of simulated botnet traffic and background traffic as shown in Figure 5.5.

Chapter 6

Machine Learning Algorithms

In this chapter, we describe the terminology to evaluate the classification results. We then give a brief description of supervised ML algorithms used in our experiments, which are applied to four different scenarios that are presented in Chapter 7.

To evaluate the results, we use the true positive (TP) rate to show the percentage of flows correctly classified as C&C traffic, the false positive (FP) rate to represent the percentage of legitimate applications flagged as botnet C&C traffic, and false negative (FN) rate to represent the percentage of botnet C&C traffic flagged as other legitimate applications. We calculate the overall accuracy of the classifier, and recall and precision rate of the C&C traffic.

- *Recall*, which is the percentage of flows in an application class or botnet C&C traffic class, i, that are correctly identified, is defined as :

$$Recall = TP_i/(TP_i + FN_i) \tag{6.1}$$

- *Precision*, which is the percentage of flows assigned to the each application class or botnet C&C traffic class, i, that are correctly is defined as:

$$Precision = TP_i/(TP_i + FP_i) \tag{6.2}$$

- *Overall accuracy*, which is the ratio of correctly classified flows to the total number flows in dataset, defined as :

$$Overall\ accuracy = \sum_{i=1}^{n} TP_i / \sum_{i=1}^{n} (TP_i + FP_i + FN_i + TN_i) \tag{6.3}$$

where n is the number of classes.

In our experiments, to perform supervised learning we used a Naive Bayes classifier, decision tree using the C4.5 algorithm, multinomial logistic regression, and logistic model trees (LMT). These classifiers are suitable for multiclass problems and are commonly applied to machine learning problems. We give a brief description of these methods below.

6.1 Naive Bayes Classifier

Naive Bayes classification is based on Bayes theorem [31]. A Naive Bayes classifier assumes that the features are conditionally independent of the class of origin, and learns a model for the class conditional probability distribution of each feature. The Bayes rule is then used to compute the class posterior probability with given features, which is used for classification.

Assume X is a vector of data samples, where each data sample, x_i, is described by m features $f_1, \ldots, f_m$. And assume now that there are k known classes, $(1, \ldots, k)$, where c_i is denoting the particular class of each data sample. With the assumption that the features are independent, the probability that unobserved data sample x with features $f_1, \ldots, f_m$ belongs to class c_i can be calculated as:

$$p(C = c_i | f_i, f_2, \ldots, f_m) = \frac{p(C = c_i) \prod_{i=1}^{m} p(f_i | C = c_i)}{p(f_1, f_2, \ldots, f_m)} \tag{6.4}$$

Then, data sample x is classified to a class c_i with maximum a posteriori decision rule as follows:

$$classify(f_1, \ldots, f_m) = \underset{c}{\operatorname{argmax}} \; p(C = c) \prod_{i=1}^{m} p(F_i = f_i | C = c) \tag{6.5}$$

In our experiments, for each feature we model with a Gaussian (normal) distribution for each class by computing the mean and standard deviation of the training data in that class.

6.2 C4.5 Classifier

The C4.5 algorithm [33] creates a decision tree, where at each node of the tree the feature(s) with normalized largest information gain is used to split the data into subgroups, ending at the leaf nodes. A decision tree should have the property that at each leaf node, a strong majority of the samples belong to one class, which is also chosen as the predicted class for samples belonging to that leaf node.

C4.5 uses two types of tests for each feature X. The test $X =?$ with a outcome is applied for discrete attributes, and it applies $X \leq \theta$ for numeric attributes where θ is a constant threshold. The candidate threshold values are specified by sorting the distinct values of X that appear in training test by obtaining a threshold between each adjacent values.

At each step of the algorithm, one feature is selected from the set of current leaf nodes with the attribute split is such that the *normalized information gain* (NIG) is greatest. With a given discrete class random variable, C, and X is the binary particular split of a given feature, normalized information gain, measured for each feature split and every leaf node, is defined as:

$$NIG(C|X) = \frac{H(C) - H(C|X)}{H(C)}, \tag{6.6}$$

where H is Shannon's entropy, and $H(C)$ is defined as:

$$H(C) = \sum_{c_i} p(C = c_i) \log_2(p(C = c_i)) \tag{6.7}$$

and $H(C|X)$ is defined as:

$$H(C|X) = -\sum_{j} p(X = x_j) \sum_{i} p(C = c_i|X = x_i) \log_2(p(C = c_i|X = x_i)) \tag{6.8}$$

To classify a given data sample, the tree starting from a root node with test nodes that have two or more splits is linked to the a subtree, goes iteratively into a subtree until it reaches the leaf nodes. The class is associated at the leaf nodes with a sufficient high class purity becomes the predicted class.

Moreover, C4.5 has many options such as error-reduced pruning, avoiding over-fitting, and dealing with missing values. In our experiments, we use subtree raising algorithm to overcome the overfitting problem. In this case, a subtree from downward may be moved upwards towards the root of the tree to replace the other subtree. Given a particular

confidence, we find confidence limits, and we use that upper confidence limit as an estimate for the error rate of the node. A error estimate for a subtree is calculated as a weighted sum of error estimates for all its leaves, and itself. If node's estimated error is less than the combined error estimate of the its leaves, they are pruned away. In our experiments, the confidence level is 0.25 and the minimum number of instances per leaf is set to 2 for pruning.

6.3 Logistic Regression Classifier

In multinomial logistic regression, we have a linear discriminant function associated with each class, and the log ratio of the probability of each class given the features, to the probability of a reference class given the features is modeled as a linear function of the features. The weights of the linear discriminant function associated with each class are determined using a maximum a *posteriori* (MAP) estimation.
Let x be a vector of a data sample $x_1, \ldots, x_n$, the probobality of data sample whether is belong to class A or not is defined as:

$$p(Y = 1|X = x) = P(x, \beta_A) \tag{6.9}$$

where $P(Y = 1|X = x)$ is the conditional probability that the data sample with features is belong to class A, and P is a function of x parametrized by the weights vector $\beta_A = (\beta_0, \ldots, \beta_n)$. Logistic regression algorithm assumes the P function as:

$$P(x, \beta_A) = \frac{e^{\beta_0 + \sum_{j=1}^{n} \beta_i x_i}}{1 + e^{\beta_0 + \sum_{j=1}^{n} \beta_i x_i}} \tag{6.10}$$

From Equation 6.10, we can write a linear function called *logit model* with given class A and features vector x:

$$log(\frac{P(x, \beta_A)}{1 - P(x, \beta_A)}) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n \tag{6.11}$$

Based on the training set, $\beta_1, \ldots, \beta n$ values should be selected for each class to classify the given data sample. The weights vector β is specified by using the maximum likelyhood estimation. With a given N data samples of feature vector $X = (X_1, \ldots, X_n)$, where

features of data sample i is $X_i = (x_1^i, \ldots, x_n^i)$. With given $Y = (y_1, \ldots, yn)$, either $y_i = 1$ if flow is belong to class A or if not $y_i = 0$.
The log-likelihood function, $L(X, \beta) = \log(p(X, \beta))$ can be written as:

$$L(X, \beta) = \sum_{j=1}^{N} [y_j \log(p(Y = 1|X_j)) + 1 - y_i \log(1 - p(Y = 1|X_j))] \tag{6.12}$$

we can get the log-likelyhood for the logistic regression by putting Equation 6.10 to $p(Y = 1|X_j)$:

$$L(X, \beta) = \sum_{i=1}^{N} [y_i \beta^T X_i - log(1 + e^{\beta^T X_i})] \tag{6.13}$$

Newton-raphson algorithm is used to find the β that maximizes the Equation 6.13 until there is no change in β.

6.4 Logistic Model Trees (LMT) Classifier

A LMT classifier learns a decision tree using the C4.5 algorithm, but the class prediction at each leaf node is done with a logistic regression model, unlike the standard decision tree, which uses voting. [36] showed that the experiments with LMT are often more accurate than C4.5 decision trees and standalone logistic regression on real-world datasets.

Chapter 7

Experimental Results

In this chapter, we evaluate how sensitive supervised classification results are to timing-based features ($goodput_{cs}$ and $goodput_{sc}$). We begin by presenting the dataset and four different scenarios, and classification results of 4 different scenarios, and overall accuracy of the classifiers are discussed.

7.1 Dataset and Experimental Scenarios

We investigated flow classification to one of ten classes, including the nine classes defined in Table 5.3 and the C&C class, *i.e.*, the C&C class was treated as a predefined class, with the classifier trained in a supervised fashion to assign flows to one of these ten classes, based on labeled training examples from all classes (including the C&C class) as shown in Figure 7.1.

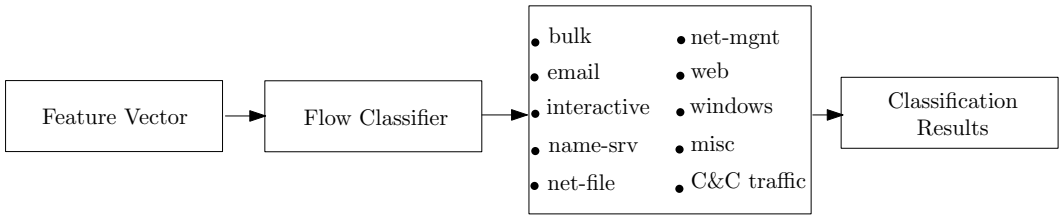

Figure 7.1. Classification Process

We salt a total of 7266 LBNL TCP flows of 9 classes described in Table 5.3 with the 50 botnet C&C traffic class samples under the following four scenarios:

(1) when the timing-based features of the botnet C&C traffic are not calibrated with the background traffic in both training and test set;

(2) when they are calibrated with the background traffic by the application of Algorithm 2 in both training and test set;

(3) when the timing-based features are not used for classification,

(4) when the timing-based features of the C&C traffic are not calibrated in the training set, but are calibrated in the test set.

Our goal is to evaluate what extent the presence of timing artifacts in botnet traces leads to changes in classifier results. For this reason, we applied Scenario (1) to show the optimistic classification results without applying the Algorithm 2 to the botnet traces. The scenario (3) reveals the classification results of exluding the timing-based features from the feature subset to evaluate whether they are sensitive for classification or not. The motivation for scenario (2) is that the attacker may deliberately arrange the timing-based features to conceal the C&C activity. Scenario (4) represents a realistic evaluation case where C&C traffic in the training set is provided by the simulated environment and tested with calibrated botnet traffic, whose timing-based features are consistent with the background traffic. This scenario begs other scenarios where the C&C traffic is not calibrated to the background traffic in the training set, and is calibrated differently in the test set in two ways: different from the background traffic or different from the C&C traffic in the training set.

The implementation of the classifiers was obtained from the open source data mining tool Weka [25]. We use stratified 10-fold cross-validation by dividing the dataset into 10 folds of approximately equal size. The proportion of classes is roughly the same in all 10 folds. We fit the model on 90% of the training data and then predict the class labels of the remaining 10% of the test data. This procedure is repeated 10 times, with each fold in turn playing the role of the test samples, and the errors on all 10 folds averaged together to compute the precision, recall and overall accuracy. Note that in creating the cross validation folds for the 4 scenarios, we use the same training and test folds, and modify only the timing-based features of the C&C traffic, where needed (*e.g.*, in scenario (4) only the timing-based features of the C&C traffic in the test folds are modified).

7.2 Classifier Evaluation

Table 7.1 shows the precision and recall rates of the C&C flows, and the average 10-fold cross-validation accuracy of each of the four classifiers. For LMT and Naive Bayes, when timing-based features are not used for classification, the C&C precision, recall rates, and

the overall accuracy all increase. For the C4.5 classifier, the results are the same in all 4 scenarios. A possible explanation is that the timing-based features are not used by the C4.5 decision tree, or the rules based on the timing-based features are not sensitive to the feature calibration. For the Naive Bayes classifier, it is interesting to note that even though the classification accuracy is not as high as the other classifiers, the C&C precision and recall rates are almost perfect. Again, the Naive Bayes classifier models the class conditional distribution of all the features and uses a Bayes rule to make class predictions. In this case, it is possible that we have an accurate model for the C&C class, which results in high precision and recall, but the model for other application classes may not be very accurate, which results in the relatively low overall accuracy. For logistic regression and LMT, the precision and recall rates are different in all the scenarios. A possible explanation is that the class posterior probability depends directly on a linear combination of all the feature values, making the predictions sensitive to the timing feature calibration.

Algorithm	Dataset	C&C Precision (%)	C&C Recall (%)	Overall Accuracy (%)
LMT	(1)	93.89	100	98.442
	(2)	89.99	98	98.057
	(3)	98.33	100	99.048
	(4)	93.89	100	98.442
C4.5	(1)	96.66	100	99.317
	(2)	96.66	100	99.317
	(3)	96.66	100	99.289
	(4)	96.66	100	99.317
Logistic Regression	(1)	67.17	98	95.381
	(2)	73.4	92	95.434
	(3)	70.63	96	95.422
	(4)	67.17	94	95.312
Naive Bayes	(1)	100	98	90.405
	(2)	100	98	90.405
	(3)	100	100	91.655
	(4)	100	98	90.405

Table 7.1. Comparison of the Classiffier C&C Precision, Recall and Overall Accuracy Rates

In Table 7.2, the number of false positives (*i.e.*, the number of legitimate applications that are classified as C&C traffic) are shown for all the scenarios. Note that for scenarios (1) and (4), since the training and test sets are the same for all application classes, the false positive counts are identical. For LMT and logistic regression, it is interesting to observe the changes in false positive counts in scenarios (1), (2), and (3). As an example, for LMT, the change from 4 to 2 for the web class, and from 1 to 3 for

	LMT				C4.5				Logistic Regression				Naive Bayes			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
web	4	2	-	4	-	-	-	-	3	2	2	3	-	-	-	-
windows	-	-	-	-	2	2	2	2	3	3	2	3	-	-	-	-
email	1	3	1	1	-	-	-	-	13	13	13	13	-	-	-	-
name-srv	-	2	-	-	-	-	-	-	7	1	5	7	-	-	-	-

Table 7.2. Comparison of the Classiffier C&C False Positive Counts

	LMT				C4.5				Logistic Regression				Naive Bayes			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
web	-	-	-	-	-	-	-	-	1	4	2	3	-	-	-	-
email	-	-	-	-	-	-	-	-	-	-	-	-	1	1	-	1
name-srv	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 7.3. Comparison of the Classiffier C&C False Negative Counts
(1) C&C traffic timing-based features are not calibrated (2) C&C traffic timing-based features are calibrated
(3) classification does not use timing-based features (4) timing-based features of the C&C traffic are not calibrated in the training set, but are calibrated in the test set

the email class; for logistic regression, the change from 3 to 2 for web class, and from 7 to 1 for the name-srv class. For C4.5 and Naive Bayes, the false positive counts are the same for all the scenarios. In Table 7.3, the number of false negatives (*i.e.*, the number of C&C flows classified as other legitimate applications) are shown. For example, for logistic regression classifier 4 C&C flows are classified as web class when timing-based features are calibrated in both the training and test sets, compared to only 1 C&C flow classified as web when the timing-based features are not calibrated.

Chapter 8

Conclusion and Future Work

In this thesis, we presented a method for salting *timing-calibrated* simulated C&C botnet traffic to the LBNL public traces. We experimented using supervised flow classification algorithms to evaluate the accuracy of detecting C&C traffic in four scenarios: (1) when the timing-based features of the botnet C&C traffic are not calibrated with the background traffic in both training and test set; (2) when they are calibrated with the background traffic by the application of Algorithm 2 in both training and test set; (3) when the timing-based features are not used for classification; and (4) when the timing-based features of the C&C traffic are not calibrated in the training set, but are calibrated in the test set. We demonstrated the changes in overall classification accuracy, C&C traffic recall, and precision rates for all scenarios. It was found that presence of any timing artifacts in botnet traces leads to changes in classifier results for some classifier models, and this could result in a serious impact on detection and false negative rates.
Our experiments yielded several insights:

First, even though we designed a classifier using the first 5 packet statistics after the 3 way handshake, we detected misclassification of botnet flows by calibrating only a few RTT samples of 2 timing-based features. Methods relying more heavily on timing-based features (*e.g.*, average inter packet spacing, flow duration) of complete flows (that consist of more RTT samples) are even more vulnerable, with smaller attack recall and precision rates. Second, by altering the network traffic patterns over time, a botmaster may change the distribution of the C&C traffic timing-based features. In this way, the botmaster may mimic the legitimate applications to disguise the botnet traffic [64]. So, we may interpret our detection results are for "worst case" calibration of timing features. Third, our experiments showed that the employed feature subset excluding the timing-

based features is well predictive of the application classes. Using calibrated timing-based features with "external" feature selection procedures for a given classification algorithm may result in more classification errors. Fourth, when porting a flow classifier from one domain to another, it is possible that domain differences in the class-conditional timing-based feature distributions may lead to substantial losses in classification accuracy on the new domain [68]. Fifth, even if the performance of some classifiers (*e.g.*, C4.5, in our experiments) are not sensitive to timing-based features, using these classifiers as part of ensemble of classifiers with a voting procedure may worsen the final decision. Finally, we suggest that to overcome the site and time dependency of RTTs, new isolated timing-based features such as application process time may result in better classification performance [27].

Although we evaluate to what extent the presence of certain timing artifacts in botnet traces leads to changes in classification results, and suggest that computer scientists repeat the methodology of this paper in their approaches. The following major points warrant elevated attention in future research:

Different types of simulated or real world botnet C&C traces (*e.g.*, P2P or HTTP based) should be considered experimentally. If the authors make the traces of recent approaches publicly available or based on request, these traces can be used to evaluate the timing-based feature perturbation of the previous approaches. However, the research community hesitates to publish such traces either because of the sensitive information they include about the involved hosts, or because of the costly process of the anonymization of large sets of packets. On the other hand, simulation of botnets has challenges of its own. First, the complete source code of the malicious botnets is not commonly available for download. Secondly, reverse engineering and analysis of prominent malicious codes of botnets requires a great amount of time, effort and attention with evolving complex structure of the bot codes [12].

In Section 5.2.5, we described our technique to calibrate the RTTs of the C&C traffic, and evaluate the datasets under four scenarios by using several supervised ML algorithms. Even if we give brief information about why classification algorithms differ in results (*e.g.*, C4.5 is invariant to our timing calibration), another interesting area would be to dig deeper into the analysis of each classifier and evaluate the how timing-based features are handled by analyzing the behaviour of each classifier when RTT calibration is performed.

Another interesting point of investigation is the attackers' capabilities of disguising the attack traffic. In our flow classifier design, we use features that are computationally

and memory-wise feasible to measure in practice, and not readily susceptible to obfuscation or tampering. In flow-based approaches, an attacker may change the distribution of the C&C traffic timing-based features or entire features by applying padding to the payload in order to change the packet sizes. However, all botnet detection techniques have their own challenges. As an example, packet-based detection is ineffective for encrypted traffic or port spoofing, DNS-based detection has its problems of identifying the agents (bots) in real time with low false negatives [10, 58]. Therefore, we think that building classifiers that ensure attackers cannot circumvent classifier accuracy which should satisfy the mechanism of correctly exporting the flows, and being tamperproof of a classifier is an important area for future research.

Appendix A

Application Classes and Sample Applications

- ftp:
  - FTP: File transfer protocol
  - HPSS: High performance storage system protocol. HPSS is designed to meet the high end of archival storage system and data management requirements
- email:
  - SMTP: Simple mail transfer protocol
  - IMAP4: Interactive mail access protocol - version 4
  - IMAP/S: IMAP4 protocol over TLS/SSL protocol
  - POP3: Post office protocol - version 3
  - POP/S: Pop3 protocol over TLS/SSL protocol
  - LDAP: Lightweight directory access protocol, it is used to access information directories from a server by email or other programs

- web:
  - HTTP: Hypertext transfer protocol
  - HTTPS: Hypertext transfer protocol with the SSL/TLS protocol
- interactive:
  - SSH: Secure shell protocol which allows data to be exchanged using a secure channel between two computers
  - telnet: Bidirectional interactive text-oriented communication protocol
  - rlogin: rlogin software utility protocol which allows users to log in another host in a network
  - X11: X window system protocol, it provides a basis for graphical user interface (GUI) for computers in a network
- name service:
  - DNS: Domain name service protocol which is utilized to identify servers by their IP addresses and aliases given their registered name
  - Netbios-NS: Netbios name service, it is a part of the NetBIOS-over-TCP protocol suite for name registration and resolution
  - Service location (SrvLoc) protocol provides a scalable framework for the discovery and selection of network services in a local area network (LAN)
- network file:
  - NFS: Network file system protocol is developed by Sun Microsystems for file sharing in UNIX systems
  - NCP: NetWare core protocol for accessing network service functions such as file accessing and messaging
- network management:
  - DHCP: Dynamic host configuration protocol
  - ident: User identity management protocol
  - NTP: Network time protocol
  - SNMP: Simple network time protocol
  - SAP: Session announcement protocol

  - NetInfo-local: Local network information protocol for displaying system IP addresses and MAC addresses

- windows:
  - CIFS/SMB: Common Internet file system/Server message block protocol, it is a network file sharing protocol where CIFS protocol is used as a dialect of SMB protocol
  - DCE/RPC: Distributed computing environment/Remote procedure call protocol, it is a specification for a remote procedure call mechanism that defines both application programming interfaces (APIs)
  - Netbios-SSN: Netbios session service protocol in Netbios API
  - Netbios-DGM: Netbios datagram service protocol in Netbios API

- miscellaneous:
  - Steltor: Calendar access protocol of steltor software
  - LPD: Line printer daemon which is used for submitting print jobs to a remote printer
  - IPP: Internet printing protocol
  - ORACLE-SQL: Oracle database listener protocol
  - MS-SQL: Microsoft SQL server protocol

Appendix B

Abbreviations

- C&C: Command and Control
- DDNS: Dynamic Domain Name System
- TTL: Time-to-live
- RTT: Round Trip Time
- DNS: Domain Name System
- RRDNS: Round Robin Domain Name System
- FFSN: Fast-flux Service Network
- DoS: Denial of Service
- DDoS: Distributed Denial of Service
- POP: Post Office Protocol
- API: Application Programming Interface
- IDS: Intrusion Detection System
- HIDS: Host-based Intrusion Detection System
- NIDS: Network-based Intrusion Detection System
- IP: Internet Protocol
- TCP: Transport Control Protocol
- UDP: User Datagram Protocol
- IPFIX: Internet Protocol Flow Information eXport
- IRC: Internet Relay Chat
- P2P: Peer-to-peer
- ISP: Internet Service Provider
- NXDOMAIN: No Such Domain
- ML: Machine Learning

Bibliography

[1] http://packetstormsecurity.org/irc/kaiten.c.

[2] Cisco IOS NetFlow configuration guide release 12.4. http://www.cisco.com.

[3] Internet Assigned Numbers Authority (IANA). http://www.iana.org/assignments/port-numbers.

[4] LBNL/ICSI enterprise tracing project. http://www.icir.org/enterprise-tracing.

[5] NetFlow Services Solutions Guide, updated. Cisco white paper.

[6] Over 1 million potential victims of botnet cyber crime. FBI Press Release, Jun. 13, 2007.

[7] Wireshark homepage. http://www.wireshark.org.

[8] The Honeynet Project. Know your enemy: Fast-Flux Service Networks. *http://www.honeynet.org/papers/ff*, 2007.

[9] P. Bacher, T. Holz, M. Kotter, and G. Wicherski. Know your enemy: Tracking botnets. *The Honeynet Project*, 2005.

[10] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *Proc. ACM CoNEXT* , 2006.

[11] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2011.

[12] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the Zeus botnet crimeware toolkit. In *Proc. Eighth Annual International Conference on Privacy Security and Trust (PST)*, 2010.

[13] H. Choi, H. Lee, H. Lee, and H. Kim. Botnet detection by monitoring group activities in DNS traffic. In *Proc. Conference on Computer and Information Technology*, 2007.

[14] B. Claise. RFC 5101 Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. *www.ietf.org*, 2008.

[15] W. Cui, Y. H. Katz, and W. tian Tan. Binder: An extrusionbased break-in detector for personal computers. In *Proc. of 2005 USENIX Annual Technical Conference*, 2005.

[16] D. Dagon, G. Gu, C. Zou, J. Grizzard, S. Dwivedi, W. Lee, and R. Lipton. A taxonomy of botnets. In *Unpublished paper*, 2005.

[17] D. Dittrich and S. Dietrich. New directions in peer-to-peer malware. In *Proc. IEEE Sarnoff Symposium*, 2008.

[18] D. Dittrich and S. Dietrich. P2P as botnet command and control: a deeper insight. In *Proc. 3rd International Conference on Malicious and Unwanted Software*, 2008.

[19] E. Florio and M. Ciubotariu. Peerbot: Catch me if you can. *Symantec Security Response, Tech. Rep*, 2007.

[20] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In *Proc. Workshop on Hot Topics in Understanding Botnets*, 2007.

[21] J. Govil and G. Jivika. Criminology of botnets and their detection and defense methods. In *Proc. IEEE International Conference on Electro/Information Technology*, 2007.

[22] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol and structure independent botnet detection. In *Proc. USENIX Security Symposium*, 2008.

[23] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through IDS-driven dialog correlation. In *Proc. USENIX Security Symposium*, 2007.

[24] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2008.

[25] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[26] T. Holz, C. Gorecki, K. Rieck, and F. Freiling. Measuring and detecting fast-flux service networks. In *Proc. Symposium on Network and Distributed System Security*, 2008.

[27] M. Jaber, R. Cascella, and C. Barakat. Can we trust the inter-packet time for traffic classification? In *Proc. IEEE International Conference on Communications (ICC)*, 2011.

[28] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for high performance. *RFC 1323, www.ietf.org*, 1992.

[29] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proc. IEEE INFOCOM*, 2004.

[30] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM SIGCOMM Computer Communication Review*, 32(3):75–88, 2002.

[31] G. H. John and P. Langley. Estimating Continuous Distributions in Bayesian Classifiers. In *Proc. Conference on Uncertainty in Artificial Intelligence*, 1995.

[32] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proc. Workshop on Hot Topics in Understanding Botnets*, 2007.

[33] R. Kohavi and R. Quinlan. Decision tree discovery. In *in Handbook of Data Mining and Knowledge Discovery*, 1999.

[34] S. Kondo and N. Sato. Botnet Traffic Detection Techniques by C&C Session Classification Using SVM. In *Proc. International Workshop on Security*, 2007.

[35] V. Krmíček and T. Plesník. Detecting botnets with NetFlow. Presentation given at FloCon Conference, Salt Lake City, UT, 2011.

[36] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1):161–205, 2005.

[37] W. Li, M. Canini, A. Moore, and R. Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *Computer Networks*, 53(6), 2009.

[38] C. Livadas, R. Walsh, D. Lapsley, and W. Strayer. Using machine learning techniques to identify botnet traffic. In *Proc. IEEE Workshop on Network Security*, 2006.

[39] G. Lu and X. Li. On the correspondency between TCP acknowledgment packet and data packet. In *Proc. ACM SIGCOMM Conference on Internet Measurement*, 2003.

[40] R. Mark, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A First Look at Modern Enterprise Traffic. In *Proc. ACM SIGCOMM Conference on Internet Measurement*, 2005.

[41] M. Masud, T. Al-khateeb, L. Khan, B. Thuraisingham, and K. Hamlen. Flow-based identification of botnet traffic by mining multiple log files. In *Proc. International Conference on Distributed Framework and Applications*, 2008.

[42] P. Mockapetris. RFC 1035 (Standard) Domain names - implementation and specification). *www.ietf.org*, 1987.

[43] A. Moore, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. *Technical Report RR-05-13, University of Cambridge*, 2005.

[44] J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In *Proc. 3rd International Conference on Malicious and Unwanted Software*, 2008.

[45] I. Nicholas and H. Aaron. Botnets as a vehicle for online crime. 2005.

[46] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. In *ACM SIGCOMM Computer Communication Review*, 2006.

[47] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 339–351, 2003.

[48] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. Fluxor: detecting and monitoring fast-flux service networks. pages 186–206, 2008.

[49] J. Quittek, T. Zseby, B. Claise, and S. Zander. RFC 3917 (Informational): Requirements for IP Flow information export: IPFIX. *www.ietf.org*, 2008.

[50] E. report. Botnets Detection, Measurement, Mitigation & Defence. *http://www.enisa.europa.eu/act/res/botnets/*, 2011.

[51] M. Roesch et al. Snort-lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*, 1999.

[52] K. Scarfone and P. Mell. Guide to intrusion detection and prevention systems (IDPS). *NIST Special Publication*, 800:94, 2007.

[53] E. Soner. Integrating passive DNS and flow data. Presentation given at FloCon Conference, New Orleans, LA, 2010.

[54] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An overview of IP flow-based intrusion detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010.

[55] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proc. IEEE Workshop on Network Security*, 2009.

[56] W. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet detection based on network behavior. *Botnet Detection*, pages 1–24, 2008.

[57] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting botnets with tight command and control. In *Proc. IEEE Conference on Local Computer Networks (LCN)*, 2006.

[58] C. su, C. Huang, and K. Chen. Fast-flux bot detection in real time. In *Proc. Recent Advances in Intrusion Detection*, 2011.

[59] B. Veal, K. Li, and D. Lowenthal. New methods for passive estimation of TCP round trip times. In *Proc. Passive and Active Network Measurement Conference*, 2005.

[60] R. Villamarín-Salomón and J. Brustoloni. Identifying botnets using anomaly detection techniques applied to DNS traffic. In *Proc. Conference on Consumer Communications and Networking*, 2008.

[61] K. Vishwanath and A. Vahdat. Swing: realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3), 2009.

[62] P. Wang, L. Wu, B. Aslam, and C. Zou. A systematic study on peer-to-peer botnets. In *Proc. 18th Internatonal Conference on Computer Communications and Networks* , 2008.

[63] M. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. Smith. Tmix: a tool for generating realistic TCP application workloads in ns-2. *ACM SIGCOMM Computer Communication Review*, 36(3):65–76, 2006.

[64] C. Wright, S. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proc. Network and Distributed Security Symposium (NDSS)*, 2009.

[65] X. Zang, A. Tangpong, G. Kesidis, and D. Miller. Botnet detection through fine flow classification. Technical Report CSE11-001, Penn State CSE Dept, Jan. 31, 2011.

[66] H. Zeidanloo, B. Manaf, P. Vahdani, F. Tabatabaei, and M. Zamani. Botnet detection based on traffic monitoring. In *Proc. International Conference on Networking and Information Technology (ICNIT)*, 2010.

[67] Z. Zhu, G. Lu, Y. Chen, Z. Fu, P. Roberts, and K. Han. Botnet research survey. In *Proc. IEEE Computer Software and Applications (COMPSAC)*, 2008.

[68] G. Zou, G. Kesidis, and D. Miller. A flow classifier with tamper-resistant features and an evaluation of its portability to new domains. In *Proc. IEEE JSAC Special Issue on Advances in Digital Forensics for Communications and Networking*, 2011.