8-2017

Using Botnet Technologies to Counteract Network Traffic Analysis

Yu Fu
Clemson University, fu2@g.clemson.edu

Using botnet technologies to counteract network traffic analysis

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Engineering

---

by
Yu Fu
August 2017

Accepted by:
Dr. Richard Brooks, Committee Chair
Dr. Pierluigi Pisu
Dr. Yingjie Lao
Dr. Yongqiang Wang
Dr. Robert Lund

Abstract

Botnets have been problematic for over a decade [87]. They are used to launch malicious activities including DDoS (Distributed-Denial-of-Service), spamming, identity theft, unauthorized bitcoin mining and malware distribution. A recent nation-wide DDoS attacks [182] caused by the Mirai botnet on 10/21/2016 involving 10s of millions of IP addresses took down Twitter, Spotify, Reddit, The New York Times, Pinterest, PayPal and other major websites.

In response to take-down campaigns by security personnel, botmasters have developed technologies to evade detection. The most widely used evasion technique is DNS fast-flux [181], where the botmaster frequently changes the mapping between domain names and IP addresses of the C&C server so that it will be too late or too costly to trace the C&C server locations. Domain names generated with Domain Generation Algorithms (DGAs) are used as the 'rendezvous' points between botmasters and bots.

This work focuses on how to apply botnet technologies (fast-flux and DGA) to counteract network traffic analysis, therefore protecting user privacy. A better understanding of botnet technologies also helps us be pro-active in defending against botnets.

First, we proposed two new DGAs using hidden Markov models (HMMs) and Probabilistic Context-Free Grammars (PCFGs) which can evade current detection methods and systems. Also, we developed two HMM-based DGA detection methods that can detect the botnet DGA-generated domain names with/without training sets. This helps security personnel understand the botnet phenomenon and develop pro-active tools to detect botnets.

Second, we developed a distributed proxy system using fast-flux to evade national censorship and surveillance. The goal is to help journalists, human right advocates and NGOs in West Africa to have a secure and free Internet. Then we developed a covert data transport protocol to transform arbitrary message into real DNS traffic. We encode the message into benign-looking domain names

generated by an HMM, which represents the statistical features of legitimate domain names. This can be used to evade Deep Packet Inspection (DPI) and protect user privacy in a two-way communication. Both applications serve as examples of applying botnet technologies to legitimate use.

Finally, we proposed a new protocol obfuscation technique by transforming arbitrary network protocol into another (Network Time Protocol and a video game protocol of Minecraft as examples) in terms of packet syntax and side-channel features (inter-packet delay and packet size). This research uses botnet technologies to help normal users have secure and private communications over the Internet.

From our botnet research, we conclude that network traffic is a malleable and artificial construct. Although existing patterns are easy to detect and characterize, they are also subject to modification and mimicry. This means that we can construct transducers to make any communication pattern look like any other communication pattern. This is neither bad nor good for security. It is a fact that we need to accept and use as best we can.

Dedication

I dedicate this dissertation to my parents and my husband Zhe Jia, who always supported my pursuit of knowledge.

Acknowledgments

First of all, I want to thank my advisor, Dr. Richard R. Brooks. It has been an honor to be your Ph.D. student. Your guidance and inspiration for the last five years motivated me to pursue new ideas and broadened my horizons of knowledge. I appreciate all of your dedications of time, ideas and funding to make my Ph.D. works possible and productive. The enthusiasm and insight you have for research mentor me throughout my pursuit of knowledge. I really treasure the opportunities that you provided for me to present in various conferences and workshops.

Second, I would like to thank Dr. Pierluigi Pisu, Dr. Yingjie Lao, Dr. Robert Lund and Dr. Yongqiang Wang for being my committee members. I appreciate the time you spent on perfecting my work and revising my dissertation. Your insightful comments made this work possible.

Third, a special thanks to my family. Words cannot express how grateful I am for your support. Your understanding and support give me the courage to pursue the Ph.D. degree. Zhe, thank you for putting up with the many quiet evenings when I was not available to talk or spend time with you because of the work.

Also, I want to thank the previous and current students in my research group. Without your help and cooperation, I will not understand some research ideas quickly. It is my pleasure to work with you. Specially, I want to thank Dr. Lu Yu, who helped me walk through ideas and pointed my mistakes.

Last but not the least, I would like to acknowledge the support of the United States State Department Department of Human Rights and Labor (grant number SLM-AQM-12GR1033, 'Internet Freedom for West Africa') and the National Science Foundation (NSF Cooperative Agreement No. 1544910, CPS:Synergy:Security of Distributed Cyber-Physical Systems with Connected Vehicle Applicatios). Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Botnets, large networks of compromised computers remotely controlled by botmasters (botherders), are responsible for a variety of malicious activities, including DDoS (Distributed-Denial-of-Service) [115], spamming, identity theft, resource theft for Bitcoin mining and malware distribution. A recent nation-wide DDoS attacks [182] involving 10s of millions of IP addresses were caused by the Mirai botnet. It was a sophisticated and highly distributed attack launched from machines infected with the Mirai botnet.

1.1 Motivations

Ever since the first botnet Eggdrop [87] appeared in 1993, many botnets have been created. The number of botnet infections is large. According to Trend Micro active botnet map [101], the number C&C servers from 10/09/2016 to 10/23/2016 is 6575, and the number of botnet connections are 2,414,840 (see Figure 1.1). Countermeasures, especially takedown operations, are not particularly effective. They destroy research honeypots and stimulate botmasters to find creative ways to hide. Also, botnets are no longer confined to PCs. Android and iOS platforms are increasingly attractive targets. Botnets evolve quickly to outwit police and security researchers.

In botnets, a botmaster sends malicious commands to bots and receives the victim information from bots by using the C&C server. The communication between the C&C server and bots is normally the weakest link in botnets. Two key technologies, fast-flux and DGA, are used to better hide the C&C server locations. Fast-flux changes the mapping between domain names and IP ad-

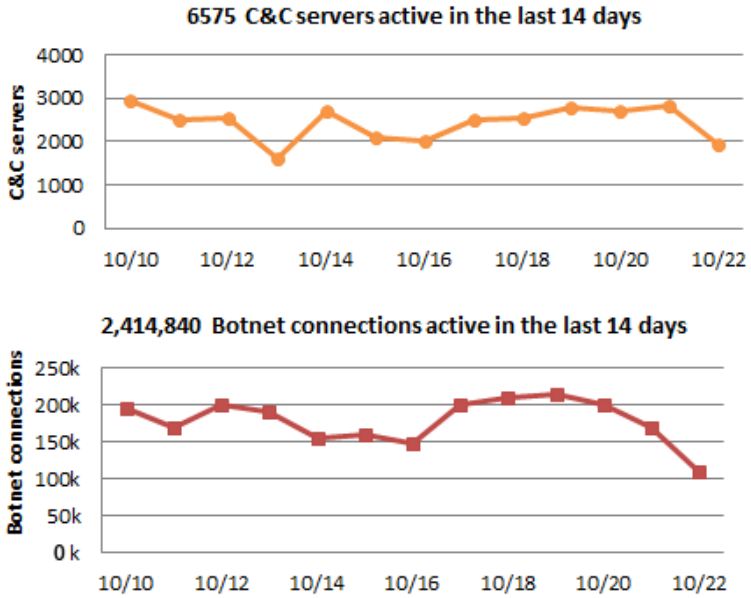


Figure 1.1: Trend Micro active botnet map [101]

dresses of the C&C server so that it will be too late or too costly to trace the C&C server locations. DGA is an algorithm to generate the large number of domain names used in fast-flux.

This work focuses on applying botnet technologies (fast-flux and DGA) to legitimate uses. We apply fast-flux and DGA to foil network analysis, especially DPI. This helps security personnel have a better understanding of the botnet phenomenon, and then develop more effective botnet detection techniques. This also allows us to be pro-active in the combat of botnets.

In addition to better understanding the criminal uses, we show beneficial applications of these concepts. The amount of transparency in the current architecture has many negative aspects, including ability to censor communications, over use of online surveillance, vulnerability to criminal abuse, and vulnerability to man in the middle (MITM) attacks.

1.2 Organizations

The organizations of the dissertation is as follows.

- Section 2 introduces the botnet phenomenon in terms of botnet economics, botnet takedown operations, botnet technologies and botnet taxonomy.
- Section 3 introduces the probabilistic models, botnet detection techniques, format-transforming

encryption (FTE), two-person zero-sum (TPZS) game and protocol tunneling related to this research.

- Section 4 proposes two novel DGAs to evade current detection techniques (KL, JI and ED). We test them against existing DGA detection systems (BotDigger and Pleiades) and conduct game theory analysis on choosing the best detection methods. Also, two HMM-based botnet DGA detection are developed to detect both existing or previously unknown DGAs.

- Section 5 applies botnet technologies to counteract network analysis by providing two applications. The first application implemented a distributed proxy system using botnet fast-flux to evade censorship and surveillance in West Africa. The second application designed a covert data transport protocol using botnet DGA to counteract network analysis. To our best knowledge, this is the first time to apply botnet technologies to real-life applications for normal Internet users.

- Section 6 develops a protocol obfuscation technique that can transform an arbitrary sensitive protocols to any target protocols. As examples, we provided two applications for offline and online transformation.

- Section 7 summarizes the dissertation. We concluded that it is possible to apply botnet techniques to protect user privacy of a normal Internet user. We can easily modify any protocol to look like another protocol. While this conclusion is new, it should not be surprising. The implications of this result deserves further study.

Chapter 2

Botnet Background

Botnets, large networks of compromised computers remotely controlled by botmasters (botherders), are responsible for a variety of malicious activities, including DDoS (Distributed-Denial-of-Service), spamming, identity theft, resource theft for Bitcoin mining and malware distribution.

During the past two decades since the first botnet Eggdrop [87] in 1993, many botnets have been created, such as GTbot (2000), Zeus (2007), Torpig (2008), etc. There are active countermeasures against botnet infestations. A research group [153] took control of Torpig for 10 days and estimated that Torpig botmasters earned between $83K and $8.3M in ten days of activities. There are also less effective countermeasures where only 24 % of botnet Command-and-Control (C&C) servers were taken down, leaving the remaining 76 % still active [47].

Fewer botnets are being created from scratch [145], while more botnet variants are being discovered. Botnet variants add functionality and target new platforms. According to Sophos 2014 [164], malware attacks on Android increased 600 percent in 12 months. In 2013, Android had 97 percent of the global share of mobile malware with 87 percent of the global smart phone market [80]. Although the increase of malware doesn't have a significant impact on the popularity of Android phones, it creates possibilities for Android botnets. We need better understanding of mobile-based botnets.

In this chapter, we start by discussing botnet economics and botnet market interaction. Then we look at botnet takedown operations as counter-measures against botnets and the botnet counter-counter-measures to understand botnets. We also present two botnet technologies: fast flux and domain generation algorithms (DGA), both of which play an important role in hiding botnet

traces. At last, we look at botnet taxonomy and detection techniques.

2.1 Botnet economics

There are multiple roles in the botnet market. They can be generalized and described briefly as:

- Botmasters: hackers that create botnet and control everything within botnet
- Command & Control (C&C) Server: prioritized nodes that distribute commands and updates to normal nodes
- Bot: normal nodes to launch malicious activities after joining botnet
- Honeypot: nodes that are controlled by security experts for research uses
- Research & Anti-virus Company: justice fighters in the botnet market
- User: common Internet user before being infected as a bot
- Government: regulating policies and controlling economics
- Bank: place where money is transferred back and forth
- Enterprise: aiming at more profits
- Mute: victim computers comprised to be mute account during dirty business

Figure 2.1 shows the interaction among these characters in botnet market [1]. After a user clicks on a bad link, malware dropper will be downloaded and executed. In this way, a user becomes a bot, which receives commands and updates from a C&C server controlled by botmasters. At the same time, the user's device will be monitored or even tampered by botmasters, especially those related to money. Through mule accounts, money can be transferred 'securely' and quietly from the user account to the botmaster account, which might be outside the country. Recent botnets use cryptocurrencies for transactions to replace mule accounts, for example, a customer can pay in Bitcoins to rent a botnet in underground dark market. In addition, botmasters use bots for spamming, malware installation, phishing & pharming, DDoS attack, bitcoin mining and so on.

Unlike the traditional seller-and-buyer market, the botnet market has the following features:

[1] Note that the price shown in the figure is based on the search results with the corresponding keywords from the Tor Darknet market search engine, Grams (06/29/2017).

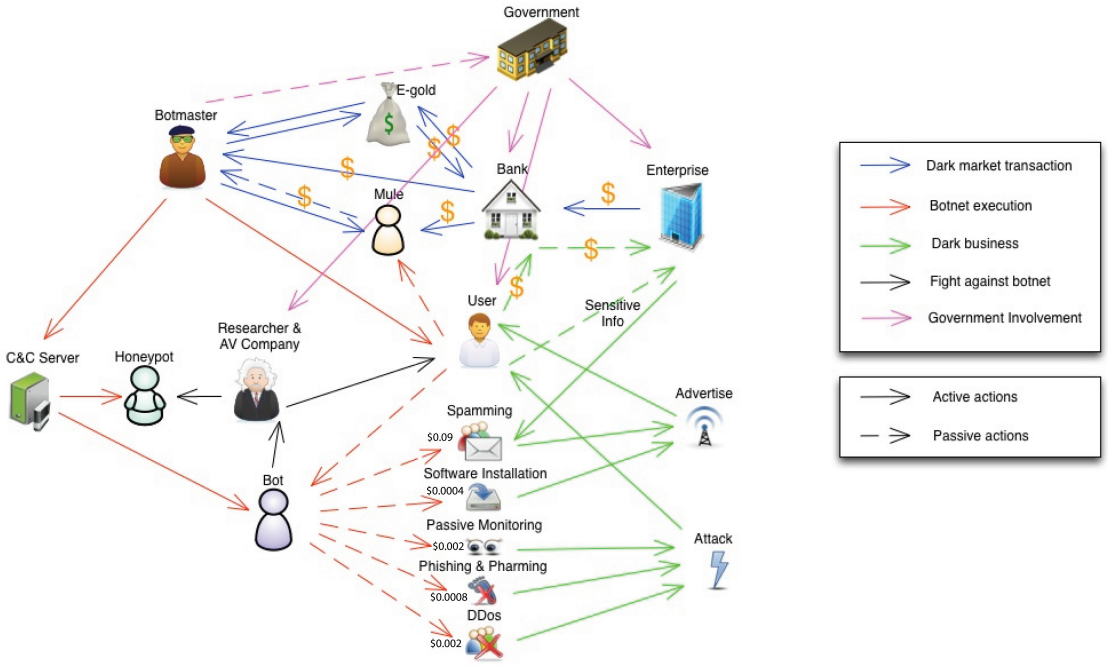


Figure 2.1: Botnet market interaction

Method of payment

According Panda Security [138], almost all botmasters prefer to receive money from their clients. The commonly used money transfer media include Liberty Reserve, Western Union, Webmoney etc. These organizations guarantee one hundred percent anonymity for botmasters and clients. Note that money transfers are being replaced by crypto-currencies. This new economic reality is changing the market and the system is still adjusting to these modifications.

Well-organized service and support

Like common online commerce, the botnet market is organized. It has ubiquitous advertisement, ranging from social media like Facebook to malware forums. These provide clear prices for services. They offer free trials and test on the 'goods' or guarantees the quality of service with data substitute if it doesn't work. They have well-designed online stores with username, password and CAPTCHA verification to prevent robot logins. Finally, after purchase, you have a control panel to launch malicious activities and most are click-based.

Quality of goods

Traditionally, the PIN number of credit cards is considered as products of good quality in dark markets. However, in botnet market, more detailed information is sold to satisfy online and offline needs, including name, date of birth, address, phone number, position of employment, driver license number, mothers maiden name of a victim etc. Some botnets guarantee to provide bots with one hundred percent online time.

Low prices

According to search results from AlphaBay market [12], it only costs $25 to rent 25k botnet to launch a HTTP-based DDoS on any website with 24-hour downtime guaranteed [167]. The seller claimes that attacks come from around 25,500 compromised systems with max 9,000 online at any given time located in 105 countries globally often with dynamic IP addresses that are changing daily. Also, one can buy the source code of the blackmail Bitcoin ransomware for $9, which is editable to steal any amount from any Bitcoin address [121]. The price of the source code of Zeus botnet is $3 and a compiled ready-to-work Zeus botnet is $70 [144]. It costs $7.5 to buy a account stealer pack which includes remote administration tools (RATs), phishing pack, keyloggers, email spoofer, cracking tools, ddos tools, Skype jacker etc. [166].

2.2 Botnet lifecycle

Generally speaking, in every botnet, each bot goes through five stages [145, 42], which we call it life cycle. After these stages are done, the victim's machine will become an active bot under the control of botmasters.

Infection As the first period of lifecycle, initial infection is always a 'trap'. It can be a tempting free software download bundled with malware, a fake update link for daily used software or unwanted email attachment. In this way, botnet toolkits are stored in the host machines and wait to be executed some time later.

Execution Secondary execution happens after initial infection. Infection programs search for a malware binary file and execute the shell code to infect the target to be a real bot. Commonly, shell

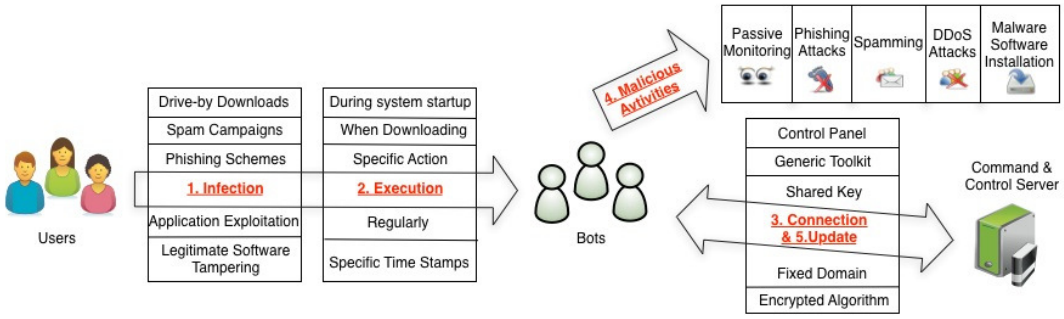


Figure 2.2: Lifecycle of botnets

code images are fetched from a specific location via FTP, HTTP or P2P protocol [42].

Connection In the connection stage, bots execute the program to initiate communication with command & control server and officially join the botnet army. This process is called rallying when a botnet client logins in to a C&C server [131].

Malicious Activities In the fourth stage, actual botnet business starts. When botmasters send malicious commands, bots are instructed to do things the host owners are not aware of, such as, spamming.

Update & Maintenance In order to keep the botnet working properly under the pressure of security polices, botmasters have to perform updates and regular maintenance. Commands and files are distributed as command malicious instructions based on the hierarchical structures of botnets.

The interaction of the five phases above is in Figure 2.2.

2.3 Botnet takedown

Security personnel often initiate takedown operations trying to remove botnets. However, some were successful, but others not. The following botnet takedown trends can be observed:

Specific botnets are taken down, only to be replaced by new and improved botnets.

When Kaspersky Lab and Microsoft took down the Kelihos botnet in 2011, Kaspersky researchers detected a new version of Kelihos in 2013 [45]. The new Kelihos botnet had better

resistance to sinkholing techniques and remained dormant longer on infected machines to evade detection. Fast-flux was introduced to hide domain names of C&C servers. The same story happened to Pushdo/Cutwail botnet. The security industry has tried to shut down Pushdo botnet four times between 2007 and 2012, and all shutdown operations resulted in temporary destruction. But in May 2013, an evolved Pushdo botnet took criminality to another level using domain fluxing as a fallback mechanism to normal C&C communication methods [118]. It seems when security experts eliminate botnets, botmasters learn from experience and make stronger and more resilient botnets.

Botnet disruption can cause more harm than good.

On June 5th, 2013, Microsoft Digital Crimes Unit worked with FBI to disrupt 1400 separate Citadel botnets, which was responsible for over $500,000,000 losses worldwide [102]. This operation was targeted at sinkholing C&C infrastructure. But 7 days later, Sophos [178] claimed that the takedown was unsuccessful after monitoring and probing sampled 72 domains, because (1) 51% of probed domains didn't appear in Microsoft published list, (2) 20% were in the list, but not sinkholed, and (3) the remaining 29% were sinkholed properly. This shows that 71% of the sampled domains are still active 7 days after the takedown operation, and implies that the domains sinkholed might be quickly re-appropriated by Citadel botnets [178] within 7 days. So regaining for this botnet is fast. Also the takedown operation might lead to false confidence and possible neglect by the public.

Takedowns destroy honeypots. Honeypots are designed to monitor cybercrooks and help identify nodes linked to ongoing frauds. When disrupted, they are unable to report IP addresses of infected clients to network operators. Other less obvious damages are also caused by botnet takedowns. In 2011, after Microsoft and FireEye Security took down Rustock botnets, spam volume dropped by nearly one third. However, an increase in emails with malicious URLs was also seen. Experts suspected that this was either a strategy of Rustock operators to recover and build larger botnets, or rivals attempting to take over the market left by Rustock [85].

Botnet takedowns prompt criminals to come up with more creative ways to do illegal business.

According to a report about Microsoft's recent actions against malware [55], when Microsoft took down Zeus botnets in 2012, it led botmasters to change the C&C structure from a centralized feedback mechanism to a peer-to-peer structure. Takedown operations often cause botnet operators

to improve their technology. In the future, we need to analyze the strategic game between botmasters, victims and security researchers to better understand criminal markets and develop strategies that limit the effects of botnets. The current approach appears to neither reduce criminality nor make the users any safer.

So botnet takedown operations can be ineffective countermeasures. However, considering takedowns are intended for instant reactions to large-scale malicious activities, consequences of takedown are understandable and warranted.

2.4 Counter-measures and botnet counter-counter-measures

2.4.1 Two-Factor-Authentication counter-counter-measures

Two-Factor-Authentication (2FA) combines two identity verifications for online activities. Traditional authentication requires one-component authentication (username and password), but 2FA adds an extra layer of security by introducing one more factor. The second factor is commonly a SMS message for online applications such as Twitter. Bank clients have to input the correct information from the SMS into online banking sessions to accomplish a transaction. The assumption is that it is unlikely that both computer and cellphone are compromised at the same time. For long, 2FA has been an effective countermeasures against cyber attacks. However, recent botnets effectively counter this countermeasure.

Three Android botnets, ZitMo (also Eurograbber, or Zeus-in-the-Mobile), SpitMo (SpyEye-in-the-Mobile) and CitMo (Carberp-in-the-Mobile) proposed counter-counter-measures against 2FA. We briefly introduce each before comparing them.

ZitMo (also Eurograbber or Zeus-in-the-Mobile)

Eurograbber, first found in 2012, is estimated to have stolen more than 36 million Euros from 30000 bank customers of multiple European banks [77]. Eurograbber starts by luring victims to click on a malicious URL and download an initial Trojan installer to their PCs. When the Trojan is installed, the victim's next login to bank accounts will be monitored. To break 2FA, the Trojan first injects JavaScript to steal the phone number, which is forwarded to the botmaster's SQL database. Victims then receive an SMS message asking them for security upgrade on cellphones. The SMS contains a link to the Android botnet installer. People using mobile phones click on phishing links

more often than those using computers [161], which reminds us of the necessity to look into BITM issues. In this way, both PC and mobile device are controlled and monitored by the botmaster. TAN (Transaction Authorization Number) for the online banking session will be intercepted, letting the botmaster transfer money without victims being aware.

SpitMo (SpyEye-in-the-Mobile)

Similar to ZitMo, SpitMo [64] is designed to break 2FA and intercept banking TANs. Differences between SpitMo and Zitmo include:

- SpitMo doesn't conceal itself as a 'security upgrade', but as a system application in Downloaded folder.
- SpitMo doesn't have user interface and does not appear in the application management window.
- SpitMo uses either SMS or HTTP send the mobile malware link. This reduces the SMS expenses.

CitMo (Carberp-in-the-Mobile)

Unlike ZitMo and SpitMo, CitMo [99] targets only Russian-speaking countries. CitMo's unique features include:

- The malware masquerades as legitimate Russian banks apps that are available in Google Play.
- The mobile malware download link is sent either via SMS or by scanning a QR-code.
- Cybergangs behind Carberp charge $2000 - $10000 for monthly subscriptions to their Trojan. They also sell targeted banking Trojans for one-time fee, $40000.

Discussion & Comparison

Several features of these BITMs are worth noting: (1) The botnet designs are probably more disturbing than their functionality; (2) Botmasters know banking procedures very well and the 2FA vulnerabilities are obvious; (3) The effectiveness and efficiency of security precautions in Google Play are questioned by these BITMs, since some of them went through the security check and were able to masquerade as legitimate applications; (4) All three BITMs take advantage of SMS interception, which is a well documented vulnerability.

2.4.2 Keylogger counter-counter-measures

Keyloggers are a common tool for compromising hosts. Keylogging software is easily available and simple to implement, which lets attackers harvest sensitive information. Verizon [39] revealed that keyloggers are now the number one threat against businesses and consumers. Notable keylogger exploits include Zeus botnet malware using keylogging to collect information for defrauding banks [114].

By attacking the communications end-point, keyloggers collect information such as passwords and cryptographic keys and short-circuit other secure communications tools. To avoid keylogging, some systems have users click on a virtual keyboard on the screen. To stop this attack, the most popular tool is using virtual keyboards and the most popular way of reinstating the attack is the use of screendumps as found in newer versions of Zeus [114].

This is another example where the initial attack (keylogging) was disabled by a technical countermeasure (displaying virtual keyboards). The botnet designers simply create a counter-counter-measure (screen dumps).

2.5 Botnet technologies

2.5.1 Fast-Flux

Fast flux, including single fast flux and double fast flux, is a common technique mostly used by botnets, which can help hide the botmaster identity (IP address) by using dynamically changed 'IP-domain name' mappings for botnet communication. The earliest record of large-scale fast-flux botnet can be traced back to the summer of 2007 [136]. Different from traditional centralized IRC botnet servers, fast-flux botnet servers cannot be taken down once and for all due to its internal complex structure. A 2012 news shows that botnets originated in Eastern Europe have stolen an estimated $78 million by using fast-flux techniques [74].

To better understand what fast flux is, we start with normal communication under DNS (Domain Name System) hierarchy. Figure 2.3 shows the basic communication steps when a user (or bot in botnet case) tries to communicate with a target (or a botmaster in botnet case) with domain name www.example.com.

(1) The user sends a query to the root domain name server (.com) with a question 'where is

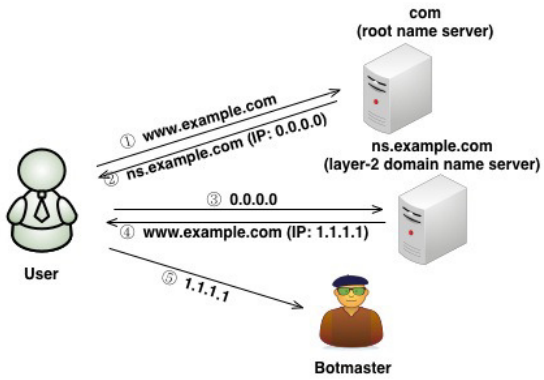


Figure 2.3: Communication under DNS hierarchy

www.example.com'.

(2) The root domain name server replies with the IP address (0.0.0.0) of layer-2 domain name server (ns.example.com).

(3) The user then queries the layer-2 domain name server with the same question 'where is www.example.com'.

(4) Layer-2 domain name server replies with the IP address (1.1.1.1) of the target.

(5) At last, the user can visit the target with the IP address.

Generally, the layer-2 domain server replies with the same or limited few IP addresses that the target domain maps to. However, in single fast flux cases, it replies with a different IP address every time a user queries and those IP addresses belong to infected mule machines within the botnet network. The mapping between the domain name and IP address is changing every 3-10 minutes based on connection quality monitoring mechanism or load-balancing algorithms combined with round-robin strategies [127]. In this case, the TTL (Time-To-Live) of an IP address is too short for security professionals to trace back the origins of the malicious activities. Even if the origin is exposed, it is going to be the mule machine rather than botmaster server. And when those intermediate victim machines are set up across different countries or using commercial DNS services [136], those bulletproofing hosting service provides [127] make it harder to expose the botmaster identities.

In double fast flux cases, not only the layer-2 domain name server replies with different IP addresses (step 4), but also the root name server replies with different layer-2 domain name server

IP addresses (step 2), which suggests that those layer-2 servers are controlled by botmasters and adds extra protection to botnet.

Since it costs little to register domain names with IP addresses, many malwares employ the idea of fast flux, such as Storm botnet [140] and a MySpace virus [139]. It can guarantee the security of criminal activities as well as make botnet infrastructure as simple as possible. Therefore, fast-flux is considered as one of the most dangerous threats to online activities today [150].

2.5.2 DGA

DGA, short for Domain Generation Algorithm, can generate a large number of domains which would be used as 'rendezvous points' between botmasters and bots. By using DGA, botmasters don't need to put a hard-coded copy of IP addresses where the C&C is or will be located. Instead, they can simply put the DGA in the bot machines and register few of the domains ahead of the communication. Every time, the bot will query the possible domains until one of them is successfully resolved to a valid IP address. The advantage of DGA is not only its capacity of securing C&C identity, but also the difficulties for the security vendors to destroy botnet since they need to register all possible domain names beforehand to sinkhole botnet communication.

For the scope of the this work, we evaluate the performance of several known DGAs, which include DGAs used by 5 live botnets (Zeus, Conficker, Kraken, Srizbi, and Torpig). We also use a research DGA (Kwyjibo) [29], because (1) Kwyjibo is one of the sophisticated DGAs, which generates words that are pronounceable yet not in the English dictionary; and (2) Yadav [181] evaluated Kwyjibo using three detection metrics (KL, ED, JI). Similar to Yadav's method, we compare our DGAs with Kwyjibo. Each DGA generates one set of malicious domain names. We briefly describe each DGA. All DGA techniques and example DGA-generated domain names are in Table 2.1.

Zeus DGA The Zeus botnet (2007) was used to steal banking information by keystroke logging and screen grabbing. We obtained pseudo code for generating domain names from the 2011 peer-to-peer variant of Zeus [5]. Domain name generation starts by taking the MD5 hash of the year, month, day, and a sequence number between 0 and 999. Then the right-most 5 bits of the MD5 hash is added to the hexadecimal value of letter 'a' to convert numbers to alphabetic characters. All generated alphabetic characters are concatenated to form a domain name.

Table 2.1: Summary of existing DGA techniques and examples

DGA	DGA techniques	Example domain names
Zeus	MD5 of the year, month, day, and a sequence number between 0 and 999	krhafeobleyhiytrwuduzlbucutwt, vsmfcabubenvibwolvgilhirvmz
Conficker	GMT date as the seed of the random number generator	fabdpdri, sfqzqigzs, whakxpvb
Kraken	a random string of 6 to 11 characters	rhxqccwdhwg, huwoyvagozu, gmkxtm
Srizbi	date transformation using XOR operations	wqpyygsq, tqdaourf, aqforugp
Torpig	current date and number 8 as the seed of the random number generator	16ah4a9ax0apra, 12ah4a6abx5apra, 3ah0a16ax0apra
Kwyjibo	Markov process on English syllables	overloadable, refillingman, multible

Conficker DGA The Conficker botnet (2008) exploited vulnerabilities of Windows OS to launch dictionary attacks. A survey [143] in 2012 found that it had infected about 7 to 15 million hosts. A research group from the University of Bonn, Germany reverse engineered DGAs of three Conficker botnet variants [84] and published domain datasets generated by the DGAs [83].

Kraken DGA Used for spamming, the Kraken botnet (2008) [30] used algorithmically generated domain names of Dynamic DNS (DDNS) providers to locate and communicate with the botnet C & C servers. To initiate communication, the malware generates a string of six to eleven characters, used as a subdomain of one of the DDNS domains (*yi.org*, *dyndns.org*, $afraid.org$, or *dynserv.com*) to form a candidate C & C domain name (e.g., $bdubefoeug.yi.org$). We did not implement the Kraken DGA, but obtained a list of 1000 bot domains [30].

Srizbi DGA The Srizbi botnet (2008) infected around 450,000 machines to send spam [112]. Given a date, the day, month, and year are split, transformed and concatenated using XOR operations. Four domain names are generated for each date throughout the year. We generated 10000 domain names using the Srizbi DGA [177].

Torpig DGA Torpig (2008) stole about 500,000 online bank accounts and credit/debit cards. We used Torpig pseudo code from [153] to generate domain names. The DGA was seeded using the current date and a constant number (8). The algorithm computed the domain name either weekly using the current week and year, or daily using the current day. We generated malicious domain names using the daily domain name generation approach.

Kwyjibo DGA The kwyjibo DGA [29] is a random word generator that guarantees the word to be short and pronounceable. It breaks dictionary words into syllables using a hyphenation algorithm [88]. It then generates random domain names based on a Markov process by making each syllable a state, and limits the length of output domain names to between two and four syllables. We use this technique [88] to obtain syllables from dictionary words for input to our DGAs. But unlike Kwyjibo, our DGAs learn from the statistical patterns from legitimate domain names and generate domain names with alphabets, numbers and '-'.

2.6 Botnet taxonomy

In the last 10 years, many botnet taxonomies have been published [195, 38, 87, 145, 9]. By generalizing and combining, five ways of classifying botnets are summarized as followed (See Table 2.2).

Table 2.2: Existing botnet taxonomy and examples

Classes	Taxonomy details / Examples			
C&C architecture [145, 9]	Centralized	Decentralized	Hybrid	Random
	Zeus [17]	Storm [162]	Hybrid P2P botnet [170]	Future botnet [28]
Communication protocol [195, 38, 87]	IRC [56]	HTTP [57]	P2P [132]	-
	Agobot [10]	Clickbot [32], Zeus/SpyEye [156]	ZeroAccess [157], Sality [155]	-
Programming language [110]	Compiled (C/C++)		Interpreted (Perl/ PHP/JavaScript)	
	Festi [100]		JavaScript Botnet [4]	
Network models [31]	Erdös-Rényi Random Graph	Watts-Strogatz Small World	Barabási-Albert Scale Free	P2P
	-	-	-	-
Features & Funtions [110]	Attack	Server	Proxy	-
	BlackEnergy [109]	Machbot [137]	ProxyBot [160]	

2.6.1 Based on Command & Control architecture [9, 145]

Centralized C&C model All bots establish communication with the one point where C&C server is located for commands and instructions. This architecture has short reaction times and good coordination.

Decentralized C&C model This model is commonly based on peer-to-peer protocol and an overlay network [69]. Since there is no central node, any loss of individual bots does not compromise botnets very significantly, leading to great flexibility and robustness of the whole network.

Hybrid C&C model Hybrid model is a combination of centralized and decentralized structure. Bots [170] are classified into servant bots (with static and routable IP and in charge of listening to connections from clients) and client bots (with dynamically designated or non-routable IP and located behind firewalls). Also, commutation within botnet is encrypted with symmetric keys, making detection difficult to implement.

Random C&C model As a future and theoretical botnet structure, bots [28] never initiate communication with botmasters, but instead, wait for connection attempts by botmasters. Zombies are found by scanning the network.

2.6.2 Based on communication protocol [195, 38, 87]

IRC protocol Short for Internet Relay Chat, IRC protocol [56] is initially designed for live interactive Internet messaging including one-to-one model and one-to-many model. When used in botnet communication, IRC can create a specific channel with standard IRC protocol, which will be authenticated by botmasters. Also, since commands through the channels can be pushed beforehand, the latency of communication is very low.

HTTP protocol Short for Hypertext Transfer Protocol, HTTP [57] is an application-level TCP/IP based protocol for distributed, collaborative, hypermedia information systems. After botmasters puts malicious commands on a web server, bots will fetch and execute them regularly from it, which is a better way to hide suspicious traffic during communication.

P2P protocol Peer-to-peer protocol [132] is the most advanced structure of botnets where each node is logically equal and there is no central server. With a known list of peers, each bot can connect with the surrounding nodes and also exchange information to update the peer list.

2.6.3 Based on programming language [110]

Compiled programming language As is known to all, compiled language is slow to develop but fast to execute. Since botmasters are technically sophisticated experts, they dont mind writing delicate codes, leading to around 80 percent of botnet codes written in C or C++.

Interpreted programming language Compared with compiled language, programming with interpreted language will be much easier but slower to execute and interpret into machine code. But there are still five percentage of botnets written in Perl.

2.6.4 Based on network models [31]

Erdös-Rényi Random Graph In this graph, botnets can be modelled as random graphs where each node has the same probability connecting to the other nodes. And the chance that a bot has a degree of k follows binomial distribution. It is found out that unless the victim is a rare high-capacity server, botnets would keep average k small, say 10.

Watts-Strogatz Small World Model In this graph, botnet network connections are created in a ring within a range of r and each bot is connected to nodes on the opposite side of the ring through a shortcut. And r is typically small, say 5.

Barabási-Albert Scale Free Model In this graph, each node has a various degree distribution following a power law, which creates a scale free structure. The network contains a small number of central and well-connected hub nodes and many leaf nodes with fewer connections.

P2P model Structured P2P network might use CHORD or CAN protocol and unstructured P2P network tends to have power-law link distribution.

2.6.5 Based on feantures and functions [110]

Attack What a botnet is capable of is mainly about attacks, such as DDoS, spamming etc. Most attacks are started from exploiting the vulnerabilities of an operating system. It is found out that botnet-based DDoS attacks are the most problematic trends in network security threats [3].

Server Server-based botnets can cover HTTP server (30%), FTP server (50%) etc. Due to the nature of a server, payload especially heavy payload will be delivered with this kind of botnet.

Proxy Proxy-based botnets generally refer to those using SOCKS to send massive information. If the botmaster enables the SOCKS proxy functionality on a remote bot, the victim machines can be an email-spamming center [65].

2.6.6 Proposed botnet taxonomy

Although multiple taxonomies are explained briefly above, each of them has advantages and shortcoming. For example, taxonomy based on Command & Control architecture classifies all botnets into centralized and decentralized, which is right but doesn't help if we want to go deeper in botnet researches. Taxonomy based on network models gives a comprehensive theoretical analysis and good mapping between graph and botnet network, but it doesn't make sense when we are exploring the mechanism of real-life botnet. Therefore, there is a necessity to propose a clear taxonomy in a different perspective. We are classifying all botnets based on their lifecycle (See Table 2.3 and Figure 2.2).

Table 2.3: Proposed botnet taxonomy based on lifecycle

Lifecycle	Taxonomy details				
1. Infection	Drive-by download	Spam campaigns	Phishing schemes	Application exploitation	Legitimate software tampering
2. Execution	During system startup	When downloading	By specific actions	Regularly	Based on time stamps
3. Connection & 5. Update	Control panel	Generic toolkit	Shared key	Fixed domain	Enverypted algorithm
4. Malicious Activities	Passive monitoring	Phishing attacks	Spamming	DDoS attacks	Malware software installation

2.7 Botnet detection

Researchers have been working on botnet detection extensively. Google scholar shows 16,500 papers with the keyword 'botnet detection'. We classify all botnet detection methods into two categories:

(1) Network anomaly based botnet detection: This cover 90 percent of botnet detection paper,

where researchers apply machine learning and data mining techniques to classify network traffic so that they can distinguish between malicious botnet traffic and normal network traffic. The same method can be applied to network intrusion detection or any other anomaly detection.

(2) Botnet specific detection: This takes advantage of botnet specific features. For example, botnet DGA can generate NXDomains, which are associated with domain query failure. This is unique to botnet only and can be applied to botnet detection effectively.

Also, there are papers about taxonomy and survey of botnet detection [163, 42, 9, 188, 79, 195, 87, 123, 54].

2.7.1 Network anomaly based botnet detection

Network anomaly based botnet detection applies machine learning and data mining techniques to distinguish botnet traffic from normal network traffic. They share similar procedures: (1) collect and label botnet traffic, (2) generate training and testing dataset from collected traffic, (3) design one or multiple features, (4) train machine learning and data mining models with the training dataset, (5) test the models with the testing dataset, and (6) calculate the detection rate and false alarm rate.

Many papers analyze network flow and detect botnet based on behavior and anomaly with clustering and classification techniques. Some works focus on applying the clustering/classification algorithm to network traffic directly. Arshad et al. [8] clustered network traffic and identified malicious hosts based on protocols and netflow features. Karasaridis et al. [78] proposed an ISP-level botnet detection method that can characterize botnets by 9 features using passive analysis based on flow data. It was performed mostly on transport layer data and didn't depend on application layer information. BotHunter [59] is a botnet detection system with correlation engine that can detect specific stages of the malware infection process, including inbound scanning, exploit usage, egg downloading, outbound bot coordination dialogue, and outbound attack propagation. Zang et al. [186] used the Hierarchical and K mean clustering algorithms to discriminate botnet traffic from normal network traffic.

Some works focus on traffic reduction before clustering and classification. Wang et al. [169] used one intrinsic traffic reduction filter, which checks DNS query/response packets and filters by associated IP addresses. Eslahi et al. [40] applied HTTP Traffic Separator (HTS) and Get

and Post Separator (GPS) to filter out HTTP traffic with GET and POST methods. Then High Access Rate (HAR) filter was used to eliminate the group of similar HTTP connections, and Low Access Rate (LAR) filter was used to remove traffic with a low access rate. Zeidanloo et al. [187] reduced irrelevant traffic by filtering out targets with whitelisted IP addresses, traffic established from external host to internal hosts, and handshaking processes. Zhang et al. [190] used a novel adaptive packet sampling algorithm and a scalable spatial-temporal flow correlation approach to substantially reduce the volume of network traffic that went through DPI to boost the scalability of existing botnet detection systems.

Some works focus on comparing the performance of different botnet detection. Stevanovic et al. [151] evaluated 8 supervised machine learning algorithm to identify botnet network traffic with 39 features and concluded that random tree classifier outperformed the rest. Feizollah et al. [43] evaluated 5 machine learning classifiers (Naive Bayes, k-nearest neighbour, decision tree, multi-layer perception, and support vector machine) and concluded that k-nearest neighbour outperformed the rest. Garcia et al. [53] compared 3 behavior-based botnet detection methods, Cooperative Adaptive Mechanism for NEtwork Protection (CAMNEP), BClus and BotHunter and developed a novel error metric for botnet detection. Abdullah et al. [2] proposed 9 criterion to evaluate botnet detection technique including unknown Botnet detection, protocol and structure independent, low false positive, low cost, low risk, encrypted bot detection, real-world detection, not require prior knowledge and reveal bot servers and C&C migration.

Although the amount of papers about network anomaly based botnet detection is hugh, they have similar drawbacks: (1) limited scope where one paper simply focuses on one or several specific botnets; (2) lack of benchmark and comparison where most papers claim they achieve almost perfect detection; (3) no shared botnet database because of the privacy issue of the background traffic; (4) limited discussion of trade-off between performance and resources (scalability) , and (5) lack of integration with existing infrastructure/hardware.

2.7.2 Botnet specific detection

Compared to network anomaly based botnet detection, botnet specific detection is more interesting. Rather than treating botnets like a blackbox, botnet specific detection takes them as a whitebox. They look at botnet specific features and conduct effective botnet detection accordingly.

Some works focus on detecting botnet DGA by looking at abnormal DNS patterns (NX-

Domains). Mowbray et al. [107] examined DNS query data to discover DGAs by identifying the unusual distribution of string lengths of domain names. Marchal et al. [98] leveraged semantic and natural language processing tools to identify malicious and dangerous domain names. Antonakakis et al. [6] proposed Pleiades, a DGA detection system that analyzes the the unsuccessful domain name resolutions (NXDomains) in DNS streams. Bilge et al. [15] developed EXPOSURE, a system that can detect malicious domain names using passive DNS analysis techniques with 15 features. Zhang et al. [189] developed BotDigger, a botnet detection system that uses 23 linguistic features and other quantity/temporal features. Choi et al. [26] proposed BotGAD, which can detect botnets using DNS traces with DNS lexical, query and answer features. Schiavoni et al. [130] designed Phoenix, a mechanism that can detect DGA-generated domain names using string and IP-based features.

Some works focus on detecting botnet fast-flux, where one domain name is mapped to multiple IP addresses. Hsu et al. [70] used passive measurement approach to detect fast-flux bots in real time, which looked at request delegation, consumer-level hardware, and uncontrollable foreground applications. Lin et al. [89] designed a Genetic-based ReAl-time DEtection (GRADE) system by employing two features: the entropy of domains of preceding nodes for all A records and the standard deviation of round trip time to all A records. Nazario et al. [111] identified over 900 fast-flux domain names with 9 features and monitored their use across the Internet to discern fast-flux botnet behaviors. Perdisci et al. [116] detected fast-flux networks based on the fact that they are accessed by users who fall victims of malicious content advertised through blog spam, instant messaging spam, social website spam and email spam.

Since drive-by download is often used as the infection vector of botnet, some works focus on detecting malware drive-by-download. Hou et al. [66] used machine learning techniques with 171 features to detect malicious web page, which is resilient to code obfuscation. Takata et al. [158] analyzed the HTTP communication of drive-by-download attacks and developed features to detect the malicious redirections. Narvaez et al. [108] showed that most anti-virus products identify less than 70% of polymorphic drive-by malware and those with successful detection fail to classify it. Hsu et al. [71] proposed a runtime, behavior-based system called BrowserGuard, to protect a browser against drive-by-download attacks.

Some works focus on detecting botnet based on malicious activities. Zhao et al. [191] implemented BotGraph to detect botnet spamming attacks targeting Hotmail users with a distributed

computing cluster. Sroufe et al. [149] presented a botnet detection mechanism by looking at the email 'shape' rather than using content or reputation analysis. Zeidanloo et al. [187] detected P2P botnets by combining malicious activities (scanning and spamming) and communication patterns. Yu et al. [185] monitored and detected online botnet activities by computing the Euclidean distance for similarity measurement and using Discrete Fourier Transform (DFT) technique.

Since network anomaly based botnet detection requires more knowledge on the legitimate network behavior, but less on the botnet designs, it might have a high false positive rate when a botnet learns from the legitimate behavior to masquerader itself and evade detection. In contrast, botnet specific detection is sensitive to botnet behaviors, but might misclassify some legitimate user behaviors. Botnets are evolving and developing 'smarter' technologies fast. Therefore, how to develop proactive botnet detection system is an important question.

Chapter 3

Mathematical background

3.1 Probabilistic models

We will introduce 3 probabilistic models (HMM, PCFG and ROC) in this section. We use HMM and PCFC to develop two new DGA algorithms. To test the anti-detection performance, we test them against existing botnet DGAs using ROC curves.

3.1.1 Hidden Markov Models (HMMs)

A Markov model is a tuple $G = (S, T, P)$ where S is a set of states of a model, T is a set of directed transitions between the states, and $P = \{p(s_i, s_j)\}$ is a probability matrix associated with transitions from state s_i to s_j such that:

$$\sum_{s_j \in S} p(s_i, s_j) = 1, \forall s_i \in S \tag{3.1}$$

A Markov model satisfies the Markov property, where the next state only depends on the current state. An HMM is a Markov model with unobservable states. A standard HMM [37, 122] has two sets of random processes: one for state transition and the other for symbol outputs. A deterministic HMM [94, 91, 133, 183] is used in this work, which only has one random processes for state transitions. Different output symbols are associated with transitions with different probability. This representations is equivalent to the standard HMM [165].

HMMs are widely used for pattern recognition and detection. Chen [91] used HMMs to

detect Zeus botnet zombies and achieved 94.7% true positive rate (tpr) and 0.7% false positive rate (fpr). Inter-packet delay of Zeus packets were collected and represented by an HMM, which was inferred with the zero-knowledge HMM inference algorithm [184]. Zhong [192] conducted a side-channel attack on Phasor Measurement Units (PMUs) in an electric power grid and used HMMs to differentiate data source in an encrypted tunnel.

3.1.2 Probabilistic Context-Free Grammars (PCFGs)

A context-free grammar (CFG) [27] is a set of recursive production rules used to generate or recognize string patterns. It is represented as a tuple $G = (N, \sum, R, S)$ in Chomsky normal form where:

(1) N is a set of nonterminal symbols, which are the placeholders for terminal symbols;

(2) $\sum$ is a set of terminal symbols, which are characters from the alphabets that the strings are made of;

(3) R is a set of production rules in the form of $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (\sum \cup N)$, which describes the recursive pattern of the grammar;

(4) S is the start symbol, and $S \in N$, which is a special nonterminal symbol that appears in the initial string generated by the grammar.

A Probabilistic Context-Free Grammar (PCFG), denoted by (G, θ), uses a probability vector θ to assign a probability to each production rule in R [19]. PCFGs are a natural stochastic model for recursive processes [96, 91]. A PCFG can also be described as a random branching process [62, 25], and its asymptotic behavior can be characterized by its branching rate. The branching rate refers to the geometric growth rate of the sentence derivation.

3.1.3 Receiver Operating Characteristic (ROC) curves

A ROC curve is a plot of false positive rate (FPR) against true positive rate (TPR) at various threshold settings. It illustrates the performance of a binary classifier system as its discrimination threshold is varied [172]. We use it to compare the detection performance of different detection metrics.

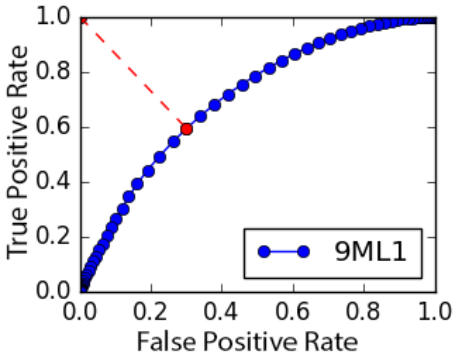


Figure 3.1: An example ROC curve

An optimal threshold is the α associated with the point on the ROC curve that is closest to (0,1). Note that point (0,1) in a ROC curve refers to perfect detection, where the false positive rate is 0 and the true positive rate is 100%. When the optimal threshold is used, it gives the optimal balance between TPR and FPR. An example ROC curve of KL detection is in Figure 3.1, where the red point is the optimal threshold, and should be used to obtain the optimal detection performance.

3.2 DGA detection techniques

In thise section, we will look at existing botnet detction techniques. Yadav [181] used Kullback-Leibler Distance, Edit Distance and Jaccard Index metrics to distinguish DGA-generated domain names (from real-life botnets such as Conficker, Torpig, Kraken etc and a research DGA called Kwyjibo [29]) from legitimate domain names. These detection schemes yield up to 100% true positive rate (TPR) and 2.5% false positive rate (FPR).

3.2.1 Kullback-Leibler distance

Kullback-Leibler (KL) Distance [81] measures information statistically. It is used in statistical language modeling [97] and information retrieval [14]. Since malicious domains are typically generated using algorithms and legitimate domains are generated manually, the letter distributions measured by KL distance are expected to differ.

For two discrete probability distributions $P = [P_1, P_2, ..., P_N]$ and $Q = [Q_1, Q_2, ..., Q_N]$,

where $\sum_{i=1}^{N} P_i = 1$ and $\sum_{i=1}^{N} Q_i = 1$, the KL distance between P and Q is:

$$D_{\mathrm{KL}}(P \mid\mid Q) = \sum_{i=1}^{N} P_i log \frac{P_i}{Q_i} \tag{3.2}$$

To make KL distance symmetric, we define $D_{\mathrm{SYM}}(P, Q)$, where

$$\begin{aligned} D_{\mathrm{SYM}}(P, Q) &= D_{\mathrm{SYM}}(Q, P) \\ &= \frac{1}{2}(D_{\mathrm{KL}}(P \mid\mid Q) + D_{\mathrm{KL}}(Q \mid\mid P)) \end{aligned} \tag{3.3}$$

We calculate the probability distribution of a domain name by counting the number of occurrences of each symbol $\theta \in \Theta$ in the domain name, where Θ is the set of symbols in a domain name (English alphabet letters and special characters). For example, the probability distribution of a domain name d is given by $P_d = \left\{ \frac{n_\theta}{|d|} : \forall \theta \in \Theta] \right\}$, where $|d|$ represents the length of d and n_θ is the number of occurrences of θ in d. For example, given $\Theta = [a, b, c, d, e]$ and $d = \mathrm{ece}$, we obtain the probability distribution of d as $P_d = [0, 0, \frac{1}{3}, 0, \frac{2}{3}]$.

To see how different a test domain d is from legitimate/malicious domain names, we calculate the average KL distance between d and a set of legitimate domain names ($\mathcal{L}$), and the average KL distance between d and a set of malicious domain names ($\mathcal{M}$). The KL distance between d and the set of legitimate domain names ($\mathcal{L}$) is:

$$\mathrm{KL}(d, \mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} D_{\mathrm{SYM}}(P_d, P_l) \tag{3.4}$$

and the KL distance between d and the set of malicious domain names ($\mathcal{M}$) is:

$$\mathrm{KL}(d, \mathcal{M}) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} D_{\mathrm{SYM}}(P_d, P_m) \tag{3.5}$$

The difference between $\mathrm{KL}(d, \mathcal{L})$ and $\mathrm{KL}(d, \mathcal{M})$ is:

$$\Delta_{\mathrm{KL}} = \mathrm{KL}(d, \mathcal{L}) - \mathrm{KL}(d, \mathcal{M}) \tag{3.6}$$

To determine if d is legitimate or malicious, we compare Δ_{KL} with a threshold value α:

$$\begin{cases} \text{if } \Delta_{\text{KL}} \geq \alpha, \quad d \text{ is malicious;} \\ \text{if } \Delta_{\text{KL}} < \alpha, \quad d \text{ is legitimate.} \end{cases}$$

We draw ROC curves [61] and vary the threshold α (from 0 to ∞) to find the optimal threshold. An optimal threshold is the α associated with the point on the ROC curve that is closest to (0,1)[1]. When the optimal threshold is used, it gives the optimal balance between TPR and FPR.

3.2.2 Edit distance

Edit Distance (ED or Levenshtein Distance) [86] inspects pairwise string alignment. It has been widely used in spell checking, plagiarism detection [154], and pronunciation measurement [63].

Edit distance gives the minimum number of single-character edits (insertion, deletion, and substitution) to transform one word to another. The computation of edit distance between two domain names d and d' is a dynamic programming problem [76], and can be broken into a collection of subproblems to calculate $lev(i, j)$, where $i = 1, \cdots, |d|$ and $j = 1, \cdots, |d'|$. Assuming $lev(i, 0) = i$ and $lev(0, j) = j$, we start with $i = 1$,$j = 1$ and alternately increment them by 1 each time until $i = |d|, j = |d'|$. Edit distance between domain names d and d' is defined as $D_{\text{ED}}(d, d') = lev(|d|, |d'|)$. For $i = 1, \cdots, |d|$ and $j = 1, \cdots, |d'|$,

$$lev(i,j) = \min \begin{cases} lev(i-1,j) + 1 \\ lev(i,j-1) + 1 \\ lev(i-1,j-1) + \begin{cases} 2, \text{ if } d(i) \neq d'(j) \\ 0, \text{ if } d(i) = d'(j) \end{cases} \end{cases} \tag{3.7}$$

where, $d(i)$ is the i^{th} character in d_i and $d'(j)$ is the j^{th} character in d'. Similar to KL distance, the difference between the edit distance from d to legitimate set $\mathcal{L}$ and the distance from d to malicious set $\mathcal{M}$ is given by:

$$\begin{aligned} \Delta_{\text{ED}} &= \text{ED}(d, \mathcal{L}) - \text{ED}(d, \mathcal{M}) \\ &= \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} D_{\text{ED}}(d, l) - \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} D_{\text{ED}}(d, m) \end{aligned} \tag{3.8}$$

[1]Note that point (0,1) in ROC curve refers to 0 false positive rate and 100% true positive rate.

We compare Δ_{ED} with a threshold value α:

$$\begin{cases} \text{if } \Delta_{\text{ED}} \geq \alpha, & d \text{ is malicious;} \\ \text{if } \Delta_{\text{ED}} < \alpha, & d \text{ is legitimate.} \end{cases}$$

3.2.3 Jaccard index

Jaccard Index (JI) [147] measures the similarity between two distributions, and is extended to spell checking and set comparison [95].

JI distance between domain name d and d' is defined as:

$$D_{\text{JI}}(d, d') = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{3.9}$$

where A and B are the sets of unique characters in domain names d and d' respectively. The assumption is that the letter distribution is different in legitimate and DGA-generated domain names. Given a test domain d, we calculate the difference between $\text{JI}(d, \mathcal{L})$ and $\text{JI}(d, \mathcal{M})$ with equation (3.10).

$$\begin{aligned} \Delta_{\text{JI}} &= \text{JI}(d, \mathcal{L}) - \text{JI}(d, \mathcal{M}) \\ &= \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} D_{\text{JI}}(d, l) - \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} D_{\text{JI}}(d, m) \end{aligned} \tag{3.10}$$

and compare it with a threshold value α:

$$\begin{cases} \text{if } \Delta_{\text{JI}} \geq \alpha, & d \text{ is malicious;} \\ \text{if } \Delta_{\text{JI}} < \alpha, & d \text{ is legitimate.} \end{cases}$$

3.3 Format-Transforming Encryption (FTE)

A network protocol is a set of rules for communication. An implicit premise most of DPI is that regular expressions are sufficient for identifying protocols [36]. Tools, such as Snort [148], Bro [106] and L7-filter [48], use regular expressions to identify network protocols.

Format-Transforming Encryption (FTE) [36] is developed to evade regular expression based protocol identification. It extends conventional symmetric encryption by formatting the ciphertext.

Arbitrary application-layer network traffic can be transformed into a target protocol using FTE. FTE is used to evade Internet censorship and surveillance, because the original protocols are obfuscated.

Although FTE is effective in hiding network protocols, it can be a challenge to write the regular expression for the target protocol. First, there is no automated software or procedures to extract regular expressions from protocols. Second, a regular expression that fully describes a protocol is usually complex. Although FTE can evade syntax-based protocol identification, it is vulnerable to side-channel analysis. Features like packet size, inter-packet delays etc., can be used for protocol identification. In this paper, a SCM method is developed to obscure side-channel features and augment FTE.

The implementation of FTE includes LibFTE and FTEproxy. LibFTE [35] is a python library to transform string to ciphertext. FTEproxy [34] is a Tor [120] pluggable transport layer to resist keyword filtering, censorship and discriminatory routing policies. LibFTE (version 0.1.0) [35] is used in this study.

3.4 Two-Person Zero-Sum (TPZS) game

Two-Person Zero-Sum (TPZS) game [168] is a classic non-cooperative finite game theory model. In TPZS, there are two players with opposite interests in which one player wins what the other player loses. The game is characterized by the strategies of each player and the payoff matrix. The payoff matrix shows the gain for both players that would result from each combination of strategies.

Suppose player 1 has sets of strategies X and player 2 has sets of strategies Y, $X \times Y$ defines the set of all situations. We define the payoff matrix as A. The TPZS is given by a triplet $\Gamma = \langle X, Y, A \rangle$. If $X = \{x_1, ..., x_m\}$ and $Y = \{y_1, ...y_n\}$, then payoff matrix is

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}, \qquad \text{where } a_{ij} = A(x_i, y_j) \tag{3.11}$$

In this form, player 1 chooses a row, and player 2 chooses a column. Since two players have conflicted interests, we suppose that player 1 wants the minimum (called minimizer) and player 2 wants the maximum (called maximizer). The goal of the game is to find the single strategy (saddle

point) or a mixed strategy that both players tend to choose as their own optimal strategy.

Note that any $m \times n$ payoff matrix can be reduced to 2×2 matrix using the dominance property [168]. Given a 2×2 TPZS game, the payoff matrix is

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \tag{3.12}$$

To solve the game, first we need to test for a saddle point. If an entry a_{ij} of the matrix A has the property that

(1) a_{ij} is the minimum of the ith row, and
(2) a_{ij} is the maximum of the jth column,

then we say a_{ij} is a saddle point. A saddle point guarantees the single optimal strategy for both players because player 1 can lose at most a_{ij} by choosing row i, and player 2 can win at least a_{ij} by choosing column j.

If there is no saddle point, the game can be solved by finding equalizing strategies. Suppose that the optimal strategy for player 1 to choose the first row has a probability of p, the probability to choose the second row is $(1 - p)$,

$$p = \frac{a_{22} - a_{21}}{(a_{11} - a_{12}) + (a_{22} - a_{21})} \tag{3.13}$$

Suppose that the optimal strategy for player 2 to choose the first column has a probability of q, the probability to choose the second column is $(1 - q)$,

$$q = \frac{a_{22} - a_{12}}{(a_{11} - a_{21}) + (a_{22} - a_{12})} \tag{3.14}$$

The expected payoff of the game is

$$V = \frac{a_{11}a_{22} - a_{12}a_{21}}{(a_{11} + a_{22}) - (a_{12} + a_{21})} \tag{3.15}$$

3.5 Protocol tunneling

Tunnelling enables the encapsulation of a packet from one protocol within the datagram of a different protocol. It allows a network user to access or provide network service that the underlying network doesn't support directly. One application of protocol tunneling is to run IPv6 protocol over IPv4 when the network doesn't support IPv6 protocol. Another application is to provide corporate service to a remote-working engineer whose IP address doesn't belong to the corporate network. In order to protect the safety of the corporate assets, the engineer can use VPN service, which can be Point-to-Point Tunneling Protocol (PPTP), Layer Two Tunneling Protocol (L2TP), or Secure Socket Tunneling Protocol (SSTP) [103].

In this work, we apply protocol tunneling to counteract network analysis. Some autocratic governments censor or block encrypted traffic [1]. We use protocol tunneling technique to tunnel SSH over DNS protocol so that any censorship targeting SSH protocol won't detect its usage. Also, DNS protocol is widely used to translate human-memorable domain names into IP address, and blocking DNS protocol can cause large-scale damage to the Internet service. So DNS protocol is unlikely being blocked by any government. Figure 3.2 shows tunneling SSH over DNS protocol.

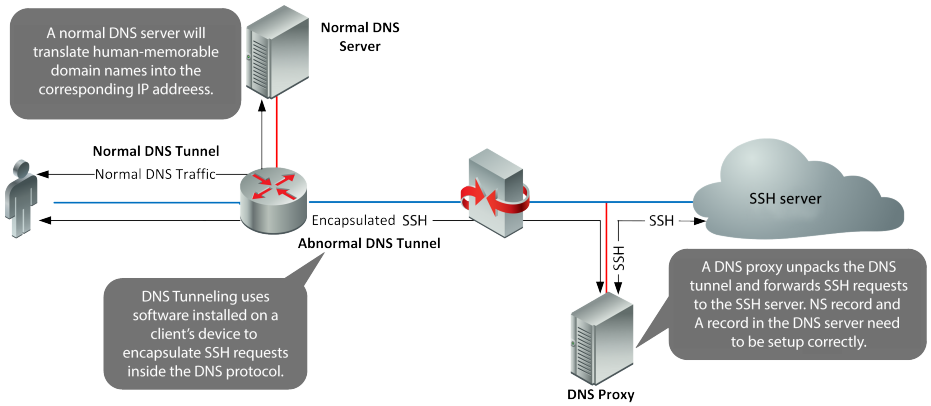


Figure 3.2: Tunnel SSH over DNS protocols

However, detecting DNS tunneling is not impossible. Statistical analysis (size of request and response and hostname entropy), uncommon record types, policy violation and traffic analysis [41] can be used to detect DNS tunneling. The idea is to identify any anomaly behavior between DNS tunneling and legitimate DNS usage.

Chapter 4

Developing DGAs and detection to understand botnets

Botnets have been problematic for over a decade [87]. They are used to launch malicious activities including DDoS (Distributed-Denial-of-Service), spamming, identity theft, unauthorized bitcoin mining and malware distribution. In response to take-down campaigns, botmasters have developed techniques to evade detection [194]. One widely used evasion technique is DNS domain fluxing [181]. Domain names generated with Domain Generation Algorithms (DGAs) are used as the 'rendezvous' points between botmasters and bots. In this way, botmasters hide the real location of command and control (C & C) servers by changing the mapping between IP addresses and domain names frequently.

Different approaches have been proposed to detect malicious domain names [179], especially DGA-generated domain names [16, 181, 124]. Yadav et al. [181] used three metrics to detect DGA generated domain names. These metrics had up to 100% detection rate and as low as a 2.5% false-positive rate. Their assumption is that domain names generated by the same DGA are similar because they are generated using the same procedure. The metrics used to differentiate between DGA and human-generated domain names are Kullback-Leibler (KL) distance, Edit distance (ED) and Jaccard Index (JI). Section 3.2 introduces these 3 detection techniques.

In section 4.1, we developed new DGAs using hidden Markov models (HMMs) and Probabilistic Context-Free Grammars (PCFGs). Since Yadav's method [181] has the best detection

accuracy among published results, we use Yadav's detection techniques to evaluate our DGAs [180]. We also benchmark different DGAs using two current DGA detection systems, BotDigger [189] and Pleiades [6]. This analysis show that our DGAs are more difficult to detect than existing ones. We model the competition between security personnel (detection techniques) and botmasters (DGAs) as a Two-Person Zero-Sum (TPZS) game. This provides a methodology for selecting optimal strategies for both sides [90], given different detection techniques and DGAs. The strategies we find should be better at detecting DGA than current approaches.

Up to now, botnet detection has been reactive, whereas botnet development has been innovative and agile. Waiting to derive new detection tools in response to criminal innovations keeps security and forensics teams at a constant disadvantage. Our new DGAs can be used to derive new pro-active tools. We use game theory to find the detection approach that should be used given current DGA detection algorithms. Our hope is that these insights can be used by security researchers to create better tools for detecting botnet infections.

In section 4.2, we developed 2 HMM-based DGA detection methods and compared them with KL, ED and JI on 2 DGA datasets. Experiment results show that HMM-based detection outperforms KL/ED/JI detection in detecting both existing DGAs and previously unknown DGAs. To our best knowledge, this is the first time that an HMM is used for DGA detection. The result is promising and helpful to improve current detection techniques.

4.1 Developing DGAs to evade botnet detection

4.1.1 DGA designs

4.1.1.1 HMM-based DGA

Our first DGA uses HMMs inferred from legitimate domain names to generate new domain names, which should have the same statistical properties as legitimate domains. The zero-knowledge HMM inference algorithm from [135, 91, 93, 184] is used to derive the HMMs. The flowchart of the HMM-based DGA is shown in Figure 4.1.

The training sets used to infer HMMs are in Table 4.1. HMMs are generated using two parameters, L and dictionary:

(1) L is the maximum number of history symbols used to calculate state transition probabilities in

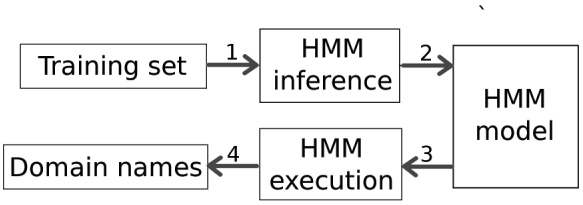


Figure 4.1: Flowchart of HMM-based DGA

Table 4.1: HMM dataset details

Name	Dictionary	L	Contain letters	Contain numbers	Pronounceable	Example Output
DNL1	English dictionary	1	yes	no	maybe	stgaarls, lohirak
DNL2		2				leaptoin, coinicaf
DNL3		3				ersonsto, bolue
DNL4		4				lyes, eftanne
9ML1	9 million Ipv4 domain names	1	yes	yes	less likely	pore9nn10, 15fapn-43f
500KL1	Randomly selected 500,000 domain names from 9ML1	1	yes	yes	less likely	b10-82025, 1-14455a
500KL2		2	yes	yes	less likely	t5154x150, 0a-1840-15
500KL3		3	yes	yes	less likely	whp01075, ota-150-pc

an HMM. Larger L produces an HMM with less conditional entropy between states. Take 'abc' as an example input string. If $L = 1$, we calculate the transition probability $P(a|a)$, $P(a|b)$, $P(a|c)$, *etc.* If $L = 2$, we calculate the transition probability $P(a|aa)$, $P(a|ab)$, $P(a|ac)$, $P(a|ba)$, $P(a|bb)$, $P(a|bc)$, $P(a|ca)$, $P(a|cb)$, $P(a|cc)$, *etc.* For our DGA, we consider transition probabilities for symbol histories of up to four letters ($L = 4$).

(2) Dictionary is the training set used to build the HMM, which is a word list containing legitimate domain names. To compare the performance of different dictionaries, we build 8 HMMs using different dictionaries and different values of L. In our experiment, three dictionaries are used:

- English word list [13];
- 9 million domain names from IPv4 space (Appendix A);
- 500,000 entries randomly selected from the 9 million IPv4 subdomains. The complexity of inferring the HMM is $O(k^{L+1}) + O(N)$ [141, 135], where k is the number of symbols, L is the maximum substring length considered, and N is the size of the input symbol sequence. As the dictionary size increases, HMM generation becomes intractable. So we use the subset of all IPv4 domains to build the HMM.

Terminals	a, c
Non-terminals	S, B
Productions	$S \rightarrow BB$ $B \rightarrow aBc\,(p = 0.25)$ $B \rightarrow \varnothing\,(p = 0.75)$

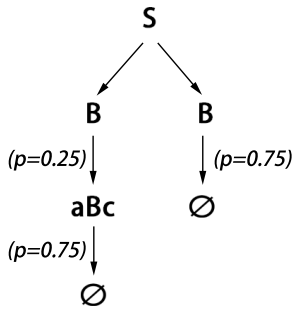


Figure 4.2: An example PCFG grammar and parse tree

Given the created HMM, we generate domain names by randomly choosing a start state in the model. Since each transition in the HMM is associated with a symbol, a transition is taken from the current state and the corresponding symbol is selected. If there are more than one possible transitions out of the current state, the transition is chosen randomly. In our experiment, each domain name is restricted to 3 to 10 letters to be consistent with the length of real IPv4 domain names. An illustrative example can be found `https://clemson.box.com/s/064mq0cpg94hg1p1f54ur0xld53ojece`.

4.1.1.2 PCFG-based DGA

PCFGs are a natural stochastic model for recursive processes [96, 91]. A PCFG can also be described as a random branching process [62, 25], and its asymptotic behavior can be characterized by its branching rate. The branching rate refers to the geometric growth rate of the sentence derivation. Since domain names must have finite length, the branching rate has to be less than 1 [62, 25].

The grammar we use is defined in Figure 4.2. This is a simple grammar, derived from parentheses matching, which is given as an example process. We use it as a simple example, because it cannot be recognized using state machines. The grammar can be visualized with a parse tree, which is a finite tree regulated by grammar rules. An example parse tree is illustrated in Figure 4.2, in which terminal a and c are two syllables randomly picked from two mutually exclusive lists of syllables A and C. A and C are random disjoint subsets of a source list (Table 4.2[1]), where $A \cup C = \textit{Source list}$ and $A \cap C = \varnothing$. For example, if $a =$ '*cr*' and $b =$ '*out*', the output of the

[1]Pronounceable refers to if the generated domain is easy to be pronounced by native English speakers.

example parse tree in Figure 4.2 would be '*crout*'. Since different source lists produce different domain names, we use 4 different source lists in our experiment:

Table 4.2: PCFG source lists

Source List	Source	Contain letters	Contain numbers	Pronoun-ceable	Example Output
pcfg_dict	English syllables	yes	no	yes	guisedswerv, daveFlorid
pcfg_dict_num	English syllables +number	yes	yes	yes	89chute, chooser3119
pcfg_ipv4	ipv4 syllables	yes	no	maybe	zenaidae, toowoom
pcfg_ipv4_num	ipv4 syllables +number	yes	yes	maybe	hrab-324, 1245ickb

- pcfg_dict: A syllable list generated from an English dictionary [13] using hyphenation [88],
- pcfg_dict_num: The syllable list in pcfg_dict plus a number list (the number list contains number from 0 to 2000, where each repeated 20 times),
- pcfg_ipv4: A syllable list parsed from IPv4 domain names (see Appendix A), and
- pcfg_ipv4_num: The syllable list in pcfg_ipv4 plus a non-alphabetic list (see Appendix B).

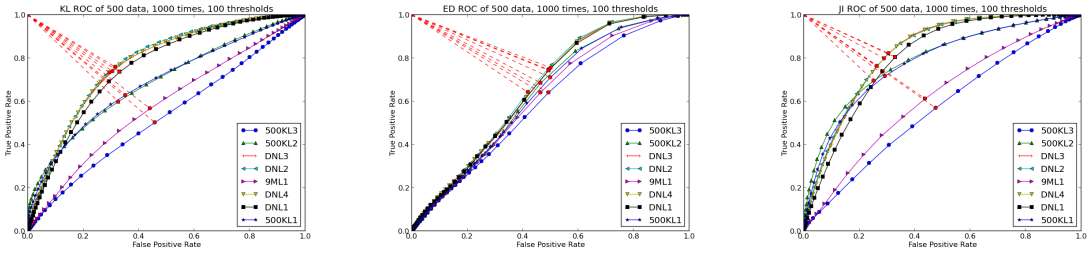
4.1.2 Experiment setup & results

For each DGA to be tested, we generate sets of domain names (test dataset $\mathcal{T}$) using this DGA. Then we compare $\mathcal{T}$ with set of legitimate domain names (legitimate dataset $\mathcal{L}$) collected using the method in Appendix A and sets of malicious domain names (malicious dataset $\mathcal{M}$) generated by the same DGA. The goal of DGA is to generate domain names that are not distinguishable from $\mathcal{L}$ and $\mathcal{M}$. The reason why we compare the DGA against itself ($\mathcal{T}$ and $\mathcal{M}$ are generated by the same DGA) is that it corresponds to the practice of security companies, where they compare each test DGA against their botnet DGA database. If there is similarity between the test DGA and existing DGAs, an alarm will be raised. So the domain names generated by the ideal DGA will not have similar character attributes, which are not the case with existing botnet DGAs.

Each test dataset contains 500 domain names. We use ROC curves to illustrate the performance of DGAs. For each threshold in ROC curves, we repeat the experiment 1000 times using different test datasets and calculate the average TPR and FPR. Each ROC curve is obtained with 100 thresholds evenly distributed from 0 to ∞ (maximum distance we observe from the data).

4.1.2.1 Test 1 - Compare different HMM-based DGAs

Figure 4.3 shows the ROC curves of HMM-based DGAs under three detection techniques (KL, ED and JI). Note that botmaster wants to maximize the distance from the ROC curve and point $(0, 1)$; or equivalently, to make the ROC curve as close to $y = x$ as possible. This means the detection cannot give a reasonable TPR while maintaining a close to 0 FPR. So the optimal DGA, or the DGA of best anti-detection performance is the one whose ROC curve is closest to $y = x$.



(a) (b) (c)

Figure 4.3: ROC curves of HMM-based DGAs under (a) KL detection, (b) ED detection, and (c) JI detection

- KL (Figure 4.3(a)): Based on the distance from the ROC curve to point $(0, 1)$, HMM-based DGAs in descending order of performance are 500KL3 > 9ML1 > 500KL1 = 500KL2 > DNL1 = DNL2 = DNL3 = DNL4 (using 95% confidence intervals). There are two reasons why 500KL3 outperforms the other HMM-based DGAs: first, the training set used to infer the HMM consists of real IPv4 domains rather than English words; and second, larger L produces an HMM with less conditional entropy between states [135], which better represents the statistical patterns in legitimate domain names. We believe the HMM inferred using all 9 million subdomains with $L = 3$ will exhibit even better anti-detection performance. However, due to the intractable computational complexity, we only provide the results of $L = 1$ (9ML1).

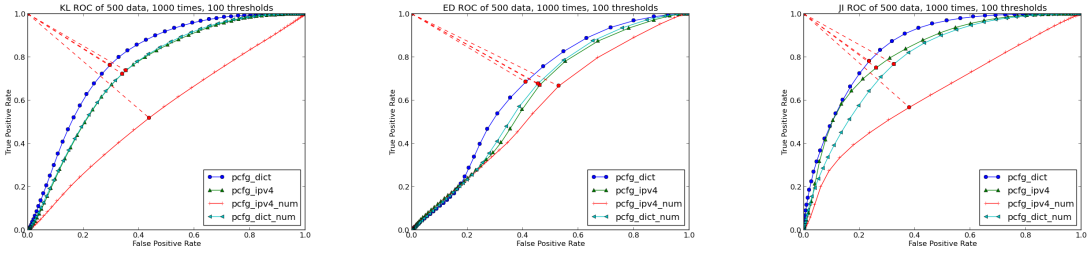
- ED (Figure 4.3(b)): All ROC curves of ED detection are close to each other and to diagonal $y = x$. So in general, ED is not an effective method for detecting domains generated with HMM-based DGAs.

- JI (Figure 4.3(c)): When JI detection is applied, the performance is 500KL3 > 9ML1 > 500KL1 = 500KL2 = DNL1 = DNL2 = DNL3 = DNL4 (using 95% confidence intervals). This is similar to the results with KL detection.

Based on the performance, 500kL3 is the best HMM-based DGA because its ROC curve is always closest to y=x under all three detection techniques.

4.1.2.2 Test 2 - Compare different PCFG-based DGAs

Figure 6.6 shows the ROC curves of 4 different PCFG-based DGAs under three detection routines:



(a) (b) (c)

Figure 4.4: ROC curves of PCFG-based DGAs under (a) KL detection, (b) ED detection, and (c) JI detection

- KL (Figure 6.6(a)): The performance is pcfg_ipv4_num > pcfg_dict_num = pcfg_ipv4 > pcfg_dict (using 95% confidence intervals). The reason why pcfg_ipv4_num outperforms the rest is: (1) the PCFG is trained using real IPv4 domains, and thus generates domains more similar to legitimate ones; and (2) by including numbers, diversity of generated domain is increased. We interpret the advantages of pcfg_ipv4_num over pcfg_dict_num as patterns in IPv4 are harder to detect than patterns in English dictionary words. That is also to say, the distance between IPv4 domain names and malicious domain names is generally smaller than that between English dictionary words and malicious domain names.

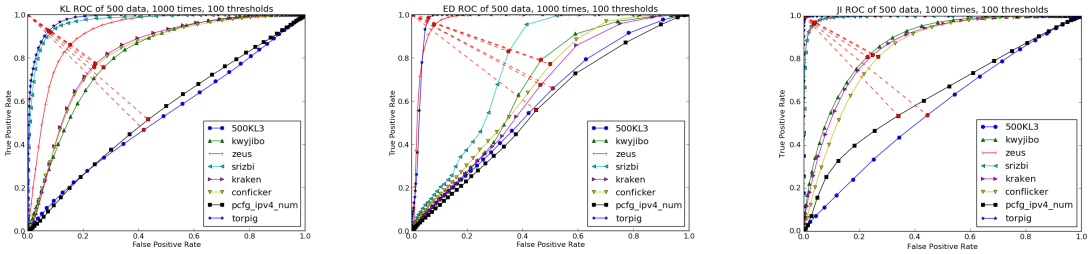
- ED (Figure 6.6(b)): Four PCFG-based DGAs show similar performance under ED detection, since all ROC curves are close to $y = x$. It indicates ED is not a reliable measurement of differences between legitimate domains and PCFG-generated domains. This is also consistent with the results of HMM-based DGAs under ED detection.

- JI (Figure 6.6(c)): The performance is pcfg_ipv4_num > pcfg_dict_num > pcfg_ipv4 > pcfg_dict. It is similar to the performance with KL detection.

Based on the performance, pcfg_ipv4_num is the best PCFG-based DGA because its ROC curve is closest to y=x under all three detection techniques.

4.1.2.3 Test 3 - Compare HMM-based, PCFG-based with Kwyjibo and 5 real-life DGAs

We compared 500KL3 (the best HMM-based DGA from Test 1), pcfg_ipv4_num (the best PCFG-based DGA from Test 2) with Kwyjibo and 5 real-life DGAs. The results are in Figure 4.5.



(a) (b) (c)

Figure 4.5: ROC curve comparison of HMM-based, PCFG-based, Kwyjibo and 5 real-life DGAs under (a) KL detection (b) ED detection (c) JI detection

- KL (Figure 4.5 (a)): The performance is: 500KL3 (HMM-based DGA) > pcfg_ipv4_num (PCFG-based DGA) >> Kwyjibo = Conficker = Kraken >> Zeus >> Srizbi = Torpig (using 95% confidence intervals). The proposed DGAs (500KL3 and pcfg_ipv4_num) are significantly better than the other DGAs, and their ROC curves are very close to $y = x$. Therefore, both proposed DGAs can effectively evade KL detection.

- ED (Figure 4.5 (b)): The performance is: pcfg_ipv4_num (PCFG-based DGA) > 500KL3 (HMM-based DGA) >> Kraken = Conficker > Kwyjibo >> Srizbi >> Zeus = Torpig (using

95% confidence intervals). Although the advantage of our DGAs are not so obvious as when KL detection is applied, they still outperform all the other DGAs. The ROC curves of our DGAs are very close to $y = x$, which indicates our approaches can effectively evade ED detection. Notice that ED is not effective in detecting HMM-based DGAs with different parameters (Figure 4.3(b)). It is the same case with detecting PCFG-based DGAs (Figure 6.6(b)). But ED is effective in differentiating HMM-based (hmm_500KL3), PCFG-based (pcfg_ipv4_num) and existing botnet DGAs. We suspect that some botnet DGAs use similar domain names (for example, Torpig uses '0ah2a2abx3apra' and '4ah0a5ax19apra'), which are close to each other in terms of ED distance. However, our DGAs don't suffer the limitations.

- JI (Figure 4.5 (c)): The performance is: 500KL3 (HMM-based DGA) > pcfg_ipv4_num (PCFG-based DGA) >> Conficker = Kraken = Kwyjibo >> Zeus = Srizbi = Torpig (using 95% confidence intervals). The advantages of HMM-based and PCFG-based DGAs over the others are obvious.

4.1.3 Result analysis

4.1.3.1 Qualitative Analysis

Based on the experiment results, it is safe to say that both HMM-based and PCFG-based DGAs thwart all three detection routines. They outperform Kwyjibo and 5 real-life DGAs. In KL/ED/JI detection, the advantages of our DGAs over Kwyjibo and 5 real-life DGAs are obvious.

4.1.3.2 Quantitative Analysis

Further analysis was carried out by modeling the competition between DGAs and detection techniques as a Two-Person Zero-Sum (TPZS) game. Although both detection techniques and DGAs in reality are not limited to those in discussion, our analysis provides a methodology for selecting the optimal strategies, given different detection techniques and DGAs. This is an example of botnet counter-counter-measurements [49].

Botmasters would like the ROC curve to be as close to $y = x$ as possible, while security personnel want to minimize the distance in the ROC curve. This allows us to model the problem as

a Two-Person Zero-Sum (TPZS) game [168] with the distance as the payoff function. Payoff function takes the distance in ROC curve from one point to the best detection point (0,1), which measures how well detectors can achieve given a threshold. The security personnel tries to minimize the payoff function, while botmaster tries to maximize it. If they end up with the same payoff function (at the saddle point), one strategy that both players tend to follow for their best interests will be obtained. Otherwise, it will result in a mixed strategy.

Table 4.3 gives the distances of different HMM-based DGA/detection pairs. Notice that maximin = minimax = 0.61. So the game has a saddle point and both players should stick to the strategy corresponding to the saddle point, i.e. 500KL3 for botmaster and ED-based detection for security personnel.

Table 4.3: The payoff matrix of player II using HMM-based DGA

		player I: security personnel (minimizer)			
		KL	ED	JI	min
player II: botmaster (maximizer)	500KL3	0.68	0.61	0.64	0.61
	500KL2	0.52	0.56	0.40	0.40
	DNL3	0.40	0.56	0.35	0.35
	DNL2	0.40	0.55	0.35	0.35
	9ML1	0.62	0.59	0.58	0.58
	DNL4	0.40	0.56	0.35	0.35
	DNL1	0.43	0.56	0.38	0.38
	500KL1	0.51	0.58	0.41	0.41
	max	0.68	0.61	0.64	

Table 4.4 gives the payoff matrix of player II (botmaster) using PCFG-based DGAs, where minimax = maximin = 0.58. Given the saddle point, the optimal strategies are pcfg_ipv4_num for botmaster and KL-based detection for security personnel.

Table 4.4: The payoff matrix of player II using PCFG-based DGA

		player I: security personnel (minimizer)			
		KL	ED	JI	min
player II: botmaster (maximizer)	pcfg_dict	0.38	0.52	0.32	0.32
	pcfg_ipv4	0.44	0.57	0.36	0.36
	pcfg_ipv4_num	0.65	0.63	0.58	0.58
	pcfg_dict_num	0.44	0.56	0.40	0.40
	max	0.65	0.63	0.58	

We also consider the scenario where the botmaster is presented with all 8 DGAs, 500KL3 (the best HMM-based DGA), pcfg_ipv4_num (the best PCFG-based DGA), Kwyjibo and 5 real-life

DGAs. The corresponding payoff matrix is shown in Table 4.5, where minimax ≠ maximin. We can solve it with an algebraic method [168].

Table 4.5: The payoff matrix of player II using HMM-based, PCFG-based, Kwyjibo and 5 real-life DGAs

		player I: security personnel (minimizer)			
		KL	ED	JI	min
	hmm_500KL3	0.68	0.61	0.64	0.61
	kwyjibo	0.37	0.51	0.30	0.30
player II:	zeus	0.21	0.09	0.05	0.05
botmaster	srizbi	0.12	0.39	0.05	0.05
(maximizer)	kraken	0.33	0.57	0.31	0.31
	conficker	0.34	0.55	0.33	0.33
	pcfg_ipv4_num	0.65	0.63	0.57	0.57
	torpig	0.11	0.06	0.006	0.006
	max	0.68	0.63	0.64	

a) First we observe that the column of KL is dominated by ED and JI. Because for each DGA, we always have $D_{KL} > min(D_{ED}, D_{JI})$. Since a security professional has no incentive to choose KL detection at all, we can delete the column of KL.

b) Similarly for the rows, we observe that the rows of hmm_500KL3 and pcfg_ipv4_num dominate the other DGAs. Since a botmaster has no incentive to choose DGAs except hmm_500KL3 and pcfg_ipv4_num, we can only keep the rows that are dominated.

c) The original matrix is reduced to a 2×2 matrix (Table 4.6).

	ED	JI
hmm_500KL3	0.61	0.64
pcfg_ipv4_num	0.63	0.57

Table 4.6: Reduced 2×2 matrix for HMM results

The optimal strategy for the botmaster is to choose hmm_500KL3 with probability p, where

$$p = \frac{0.57 - 0.63}{(0.61 + 0.57) - (0.63 + 0.64)} = 0.67 \tag{4.1}$$

The optimal strategy for the security personnel is to choose ED detection with probability q, where

$$q = \frac{0.57 - 0.64}{(0.61 + 0.57) - (0.63 + 0.64)} = 0.78 \tag{4.2}$$

Table 4.7: Collision percentages between DGAs and legitimate domain names

DGA	hmm _500KL3	pcfg _ipv4_num	kwyjibo	conficker
Collision Percentages	3.9	5.2	7.7	0
DGA	zeus	srizbi	kraken	torpig
Collision Percentages	0	0.1	0.5	0

And the expected value of the game yields

$$V = \frac{0.61 \times 0.57 - 0.63 \times 0.64}{(0.61 + 0.57) - (0.63 + 0.64)} = 0.62 \tag{4.3}$$

where V is the expected distance between the ROC curve and point (0,1), when both players stick to their optimal strategy.

4.1.3.3 Discussion

For botnet DGAs, properties other than detection evasion must be considered. One is the collision of generated domain names with existing (registered) benign domain names. Because botmasters prefer benign-looking, but not registered domain names for C&C servers. We investigate the collision rate between all DGA-generated domains (our best HMM-based DGA, PCFG-based DGA, Kwyjibo and 5 real-life botnet DGAs) and legitimate domain names. Figure 4.7 shows the percentage of domain name collisions between DGAs and benign domain names. We see that our DGAs have at most a 5% collision rate. The research DGA Kwyjibo has an 8% collision rate. The real-life DGAs have almost no collisions (less than 1%), which suggests that they sacrifice their ability to evade detection for fewer collisions. Or, alternatively, they do not mimic legitimate domain names well enough to get collisions. Also, since DGAs are used to generate domain names as rendezvous points for botmasters and bots, they want to generate the same domain name within a reasonable, finite number of iterations. For HMM-based and PCFG-based DGAs, this is possible. When botmasters and bots agree on the starting state and seed of the random number generator, the path in HMM and PCFG is guaranteed to be the same, so the generated domain names are the same. These DGAs could be deployed in a manner that reduces the number of NXDomain responses.

An HMM is a probabilistic regular grammar, and a PCFG is a context-free grammar. According to the Chomsky Hierarchy [27], unlike regular grammars, context-free grammars can

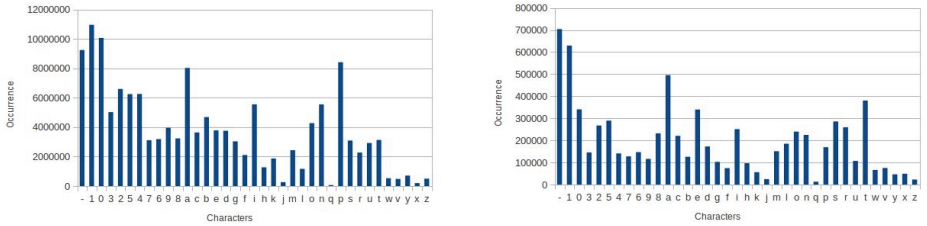
generate recursive patterns. So in theory, the PCFG-based DGA will produce patterns that cannot be expressed by regular expressions. These patterns should be harder to detect than HMM-based approaches. For the three approaches used:

- KL distance looks at letter probability distributions. These patterns exist in both HMMs and PCFGs. Whereas the context-free patterns could include more complex internal (e.g., recursive) patterns.
- Edit distance looks at changes of individual letters between domain names. Since regular grammars construct names letter by letter, it is reasonable for this approach to detect these patterns. But context-free grammars can embed sequences of letters between two letters, so one would expect ED to be less effective.
- JI, like KL, is based on the probability distributions of letters. Both regular and context-free grammars should produce characteristic letter distributions. Once again, the context-free grammars could have more complex internal structures.

The existing detection approaches look at probability distributions for letters and letter pairs, which are features suited to regular grammars. It should be possible to design PCFGs that manipulate these features to avoid detection. Research into more complex PCFGs should be fruitful.

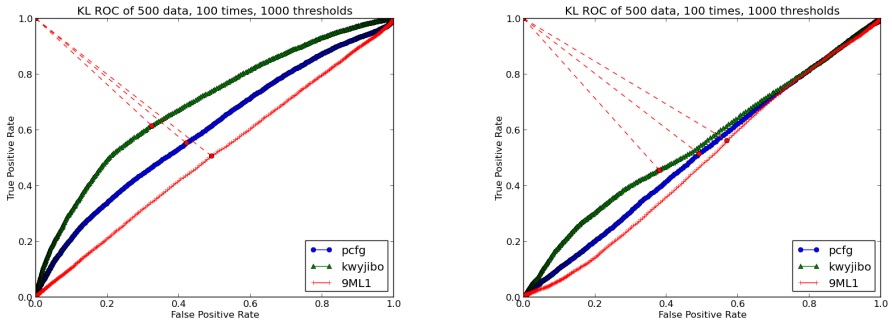
The ROC curves in KL/ED/JI detection give us similar results in detecting HMM-generated and PCFG-generated domain namesin detecting HMM-generated and PCFG-generated domain names. This is because, the parse tree (Figure 4.2) and dictionaries used for the PCFGs (Table 4.2) may not be ideal as PCFG inputs. It is difficult to find a dictionary with letters and numbers to represent the characteristic patterns in real IPv4 domain names. More research is needed to find the effective context-free techniques for generating effective DGAs. Questions related to proper dictionary development are also interesting directions for future work.

The collected legitimate domain names from the entire IPv4 space were pre-processed (Appendix A) before inputting them as training sets to infer HMMs. Statistics of characters in domain names with and without pre-processing are in Figure 4.6. The figure shows significant difference in the symbol distributions. To prove that the performance of our DGAs are not affected by the preprocessing technique, we compare the performance of DGAs generated by two datasets. Figure 4.7 (a) shows the performance of DGAs generated without pre-processing under KL detection, and Figure 4.7 (b) shows the performance with pre-processing. We can see that there is a consistent



(a) (b)

Figure 4.6: Statistics of characters in domain names (a) without pre-processing (b) with pre-processing



(a) (b)

Figure 4.7: (a) KL detection on training sets without pre-processing (b) KL detection on training sets with pre-processing

tendency HMM > PCFG > Kwyjibo, and both HMM and PCFG are very close to $y = x$. This shows that HMM and PCFG approaches are resilient. Our pre-processing has some influence, but it is not essential.

With each new defense (attack), the botnets (network defenders) adapt. The fight against network criminality has become an endless game of cat and mouse. One could suppose that network defenders could use HMMs/PCFGs to detect our new DGAs. The inferred HMMs/PCFGs represent the statistical features of both legitimate and HMM/PCFG-generated domain names, which means they are identical statistically. If they use the HMMs/PCFGs to distinguish our DGA-generated domain names from legitimate domain names, it will have a high FPR. Network defenders would have to combine these lexical features with other features, like NXDomain returns, for detecting DGA use. However, it is reasonable to use HMMs/PCFGs to detect botnet DGA-generated domain names, which would be a promising future work.

4.1.4 Testing against existing DGA detection systems

In this section, we evaluated the DGAs discussed earlier (our DGAs and botnet DGAs) against two DGA-detecting systems, BotDigger [189] and Pleiades [6]. Both claim that they can achieve up to 100% TPR and 0 FPR for detecting botnets using DGA-generated domain names.

4.1.4.1 BotDigger

BotDigger [189], proposed by Zhang et. al in 2016, detects botnet DGAs. Botnets typically generate multiple domain names, but only pay to register a few. This means that DGA traffic is correlated with many Non-Existent Domains (NXDomains) returns. BotDigger uses NXDomain correlation together with features including quantity, temporal and linguistic evidence to detect an individual bot by monitoring traffic in local network. Their experiment shows that BotDigger can detect all Kraken bots and 99.8% of Conficker bots. Since our work focuses on linguistic features, we can test the DGAs using the linguistic features processing of BotDigger. The other features are outside the scope of this work. The BotDigger code is available online (`https://github.com/hanzhang0116/BotDigger`) and we re-designed the experiments to test our DGAs using this code.

We use the 11 domain linguistic features from BotDiggers to detect DGAs:

#1: length of dictionary words in a domain name

#2: percentage of dictionary words in a domain name

#3: length of the longest meaningful substring (LMS) in a domain name

#4: percent of the length of the LMS in a domain name

#5: entropy in a domain name

#6: normalized entropy in a domain name

#7: number of distinct digital characters in a domain name

#8: percent of distinct digital characters in a domain name

#9: number of distinct characters in a domain name

#10: percent of distinct characters in a domain name

#11: length of a domain name

To calculate the dissimilarity of a domain linguistic feature between two domains (denoted as D_1 and D_2), we denote the features of D_1 as f_{1j} and the features of D_2 as f_{2j} where $1 \leq j \leq 11$. Based on the paper [189], the dissimilarity of feature j is calculated as:

$$S_j(D_1, D_2) = \begin{cases} 0, & \text{if } f_{1j} = 0 \text{ and } f_{2j} = 0. \\ \frac{|f_{1j} - f_{2j}|}{max(f_{1j}, f_{2j})}, & \text{otherwise.} \end{cases} \tag{4.4}$$

The overall dissimilarity of all features between D_1 and D_2 is calculated as:

$$S_{All}(D_1, D_2) = \sqrt{\frac{\sum_{j=1}^{11} S_j(D_1, D_2)^2}{11}} \tag{4.5}$$

After all 11 features are extracted from all domain names, hierarchical clustering algorithm is used to classify the domain names. In one experiment and for each DGA, we mix 100 DGA-generated domain names with 100 legitimate domain names (subdomain names randomly chosen from the same domain name), and classify them with the hierarchical clustering algorithm. We start with taking each domain name as a cluster and combine the two clusters with the least dissimilarity until there are two clusters left. One cluster is labeled as 'malicious' and the other one is 'legitimate'. Then a set of TPR and FPR can be calculated based on the result. We repeat the experiments 1000 times and calculate the mean and standard deviation (Sd) of all TPRs and FPRs, and the distance between (mean_FPR, mean_TPR) and (0,1). The result is in Table 4.8 and Figure 4.8[2].

From the distance in Table 4.8 and Figure 4.8, we can see that the performance of DGAs in evading BotDigger is hmm_500KL3 > pcfg_ipv4_num > torpig > kraken = zeus > srizbi > conficker = kwyjibo (using 95% confidence intervals). The results show that our DGAs can evade the cutting-edge DGA-detecting systems. We suspect the reasons why BotDigger is not effective are because (1) linguistically, our DGAs are more advanced in mimicking legitimate domain names than other botnet DGAs, and (2) for botDigger, other features like quantity and temporal evidence are important to detect DGAs, even though they are outside the scope of this work. The DGAs we provide should be more difficult for BotDigger to detect than other DGAs. We suspect that BotDigger performance in detecting our DGAs would be improved if the other features were available for evaluation. Possibly,

[2]Note that we might use the complements of the decisions since these choices provide better results than the original choices. When ROC curves are consistently below the diagonal, it is common practice to use the complement.

Table 4.8: The mean and standard deviation of the TPR and FPR, and the distance between (mean_FPR, mean_TPR) and (0,1) under BotDigger detection

DGA	Mean of TPR	Mean of FPR	Sd of TPR	Sd of FPR	distance to (0,1)
pcfg_ipv4_num	0.89	0.96	0.25	0.13	0.89
hmm_500KL3	0.92	0.97	0.50	0.16	0.92
kraken	0.56	0.94	0.49	0.15	0.57
zeus	0.56	0.92	0.50	0.18	0.57
conficker	0.45	0.93	0.50	0.16	0.46
torpig	0.75	0.91	0.43	0.19	0.75
srizbi	0.48	0.92	0.50	0.18	0.49
kwyjibo	0.42	0.90	0.49	0.20	0.44

BotDigger could be augmented to use more advanced lexical features.

4.1.4.2 Pleiades

Pleiades [6], proposed by Antonakakis et.al in 2012, also uses NXDomains of DNS traffic to differentiate DGA-generated domain names and legitimate domain names, and achieves 99% TPR and 0.1% FPR in detecting Conficker DGA. Since Pleiades is commercialized from Damballa, we re-write the code based on the paper [6].

We use 19 features from Pleiades to detect DGAs:

#1-12: The median, mean and standard deviation of the distribution of n-gram frequency from a list of domain names, where n = 1,2,3,4

#13-14: The mean and standard deviation of the Shannon entropy of a list of domain names

#15-18: The mean, median, standard deviation and variance of the length of a list of domain names

#19: The number of distinct characters appeared in a list of domain names

In one experiment and for each DGA, we have 100 groups of domain names, half of which are DGA-generated domain names and the other half are legitimate domain names (subdomain names randomly chosen from the same domain name). Each group contains 10 domain names. We apply k-means clustering (k = 2) to the 100 groups and get 2 clusters. One cluster is labelled as 'malicious' and the other one is 'legitimate'. As with BotDigger, we repeat the experiments 1000 times and calculate the mean and standard deviation of all TPRs and FPRs, and the distance between (mean_FPR, mean_TPR) and (0,1). The result is in Table 4.9 and Figure 4.9.

Table 4.9: The mean and standard deviation of the TPR and FPR, and the distance between (mean_FPR, mean_TPR) and (0,1) under Pleiades detection

DGA	Mean of TPR	Mean of FPR	Sd of TPR	Sd of FPR	distance to (0,1)
pcfg_ipv4_num	0.114	0.014	0.32	0.12	0.89
hmm_500KL3	0.192	0	0.39	0	0.81
kraken	0.185	0	0.39	0	0.82
zeus	0.297	0	0.46	0	0.70
conficker	0.216	0	0.41	0	0.78
torpig	0.218	0	0.41	0	0.78
srizbi	0.227	0	0.42	0	0.77
kwyjibo	0.197	0	0.40	0	0.80

From the distance in Table 4.9 and Figure 4.9, we can see that the performance of DGAs in evading Pleiades is pcfg_ipv4_num > kraken = hmm_500KL3 = kwyjibo = conficker = torpig = srizbi > zeus (using 95% confidence intervals). From the TPR and FPR, Pleiades detection of all DGAs achieve less than 29.7% TPR and almost 0 FPR. This shows that the linguistic features alone are not effective for detecting DGA use. Note that the result is the best scenario case for Pleiades, because we put domain names generated by the same DGA into the same group before the clustering. In real-life DGA detection, this is the best case assumption. We suspect the reasons why Pleiades is not effective are because (1) the premise of Pleiades is that DGA generates large number of NXDomain names, which need not necessarily be true, and (2) the original Pleiades take information from the second-level domain names, that is to say, statistically similar third-level domain names under the same second-level tend to belong to the same cluster, which will not work well if we consider the third-level domain names alone. The result shows the possibility of designing 'perfect' DGA that can evade Pleiades. Pleiades success seems to depend on leveraging the NXDomain features. We believe both Pleiades and BotDigger could benefit from integrating more sophisticated linguistic primitives.

4.1.5 Summary

In this section, we propose two novel DGAs (HMM-based and PCFG-based) and compare them with five real-life botnet DGAs and the Kwyjibo DGA. Three detection schemes (KL, ED, and JI) are used, which have been proven effective [181] in detecting DGA-generated domains. Experimental results show that the proposed DGAs not only successfully thwart these detection techniques, but also outperform the other DGAs in terms of anti-detection performance. The two

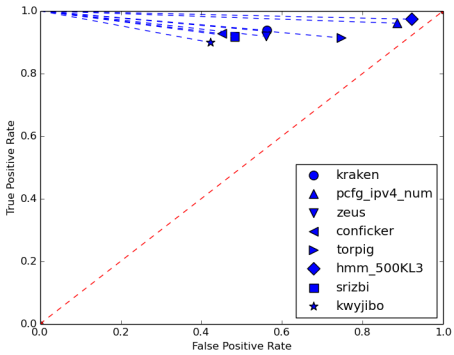


Figure 4.8: ROC curves of all DGAs under BotDigger detection system

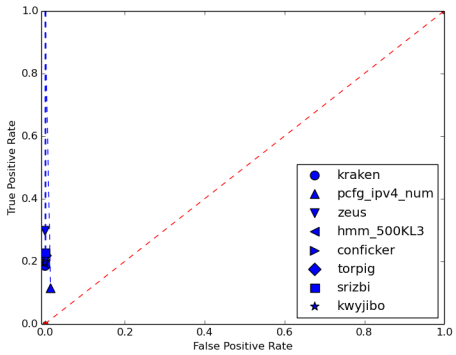


Figure 4.9: ROC curves of all DGAs under Pleiades detection system

DGAs also evade deployed botnet detection systems, BotDigger and Pleiades. Game theory analysis is used to find the optimal strategies for security personnel and botmasters. We propose future improvements and research topics which might improve our DGAs as well as DGA detection tools. The use of game theory for designing DGA detection systems seems particularly promising. We hope this work will help security personnel choose the optimal detection techniques to detect C&C servers and finally take down the botnets.

4.2 Detecting DGA-based botnets using HMMS

4.2.1 HMM-based detection designs

In our previous work [51], 8 HMMs were inferred from the IPv4 domain names. Among them, the HMM inferred from 9 million IPv4 domain names ($L = 1$) outperformed the others. The domain names it generated were indistinguishable from legitimate domain names using the lexical DGA detection approaches in [181]. In this work, we use that HMM (shown in Figure 4.10) for DGA detection (see [51] for a detailed discussion of how this HMM is inferred). Since the HMM expresses the statistical features of the legitimate domain names, we can find the corresponding Viterbi path [46] of a given domain name, which indicates the likelihood that the domain names is generated by the HMM. The probability returned by the HMM is a measure of how consistent the domain name is with the set of legitimate domain names.

[3]Note that the HMM used is inferred with $L = 1$. When $L = 1$, all transitions with the same symbol go to the same state [142]. We find the starting state by checking the first character, which reduces the number of iterations significantly.

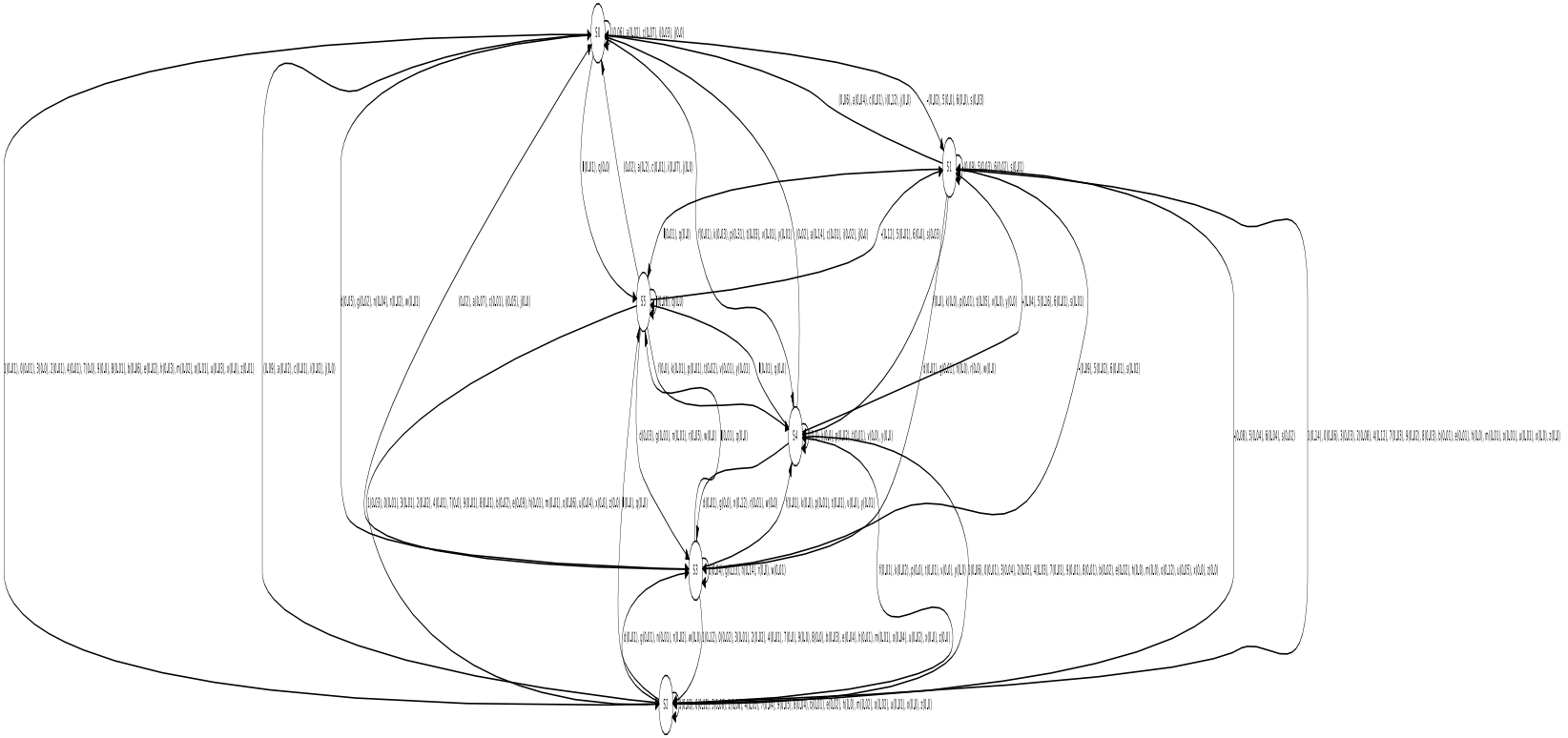


Figure 4.10: The HMM inferred from 9 million legitimate domain names

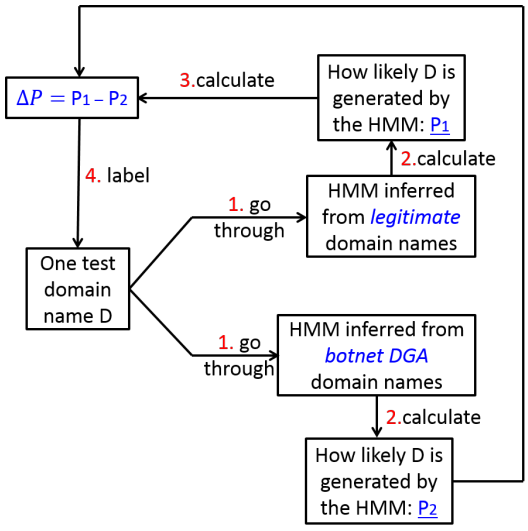
Data: A domain name D with n characters: $[d_1, d_2, ..., d_n]$, and an HMM
Result: A normalized transition probability P that D is generated by the HMM

Step 1: Find the state S for ${d_1}^3$, and p_1 is the maximum transition probability associated with d_1;
Step 2: Starting from state S, find the paths of $[d_2, ..., d_n]$ and get the associated transition probabilities $[p_2, ..., p_n]$;
Initialization $i = 2$, $flag = 0$;
while $i \leq n$ **do**
    if d_i *is in the outgoing transitions of state* S **then**
        get the transition probability as p_i;
        replace S with the next state;
        $i = i + 1$;
    else
        $flag = 1$;
        break;
    end
end
if *flag == 0* **then**
    $P = \sum_{i=1}^{n} p_i/n$;
else
    $P = 0$
end

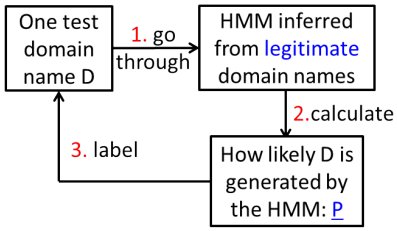
Algorithm 1: Calculate the normalized probability that a domain name D is generated by an HMM

We use a modified Viterbi algorithm [46] (shown in Algorithm 1) to calculate the normalized probability that a domain name is generated by an HMM. The original Viterbi algorithm calculates the product of all transition probabilities on the Viterbi path. In our algorithm, we use the average transition probability for several reasons:

- Most transition probabilities in our HMM are around 0.01 and the smallest can be 8.0×10^{-4}. The product of transition probabilities easily causes an underflow.
- Domain names have different lengths. Using the original Viterbi algorithm, a longer domain name forcibly has a smaller probability. To have a fair comparison, we use the average probability [94, 91, 184].
- In the original Viterbi algorithm, one smaller transition probability can lead to a smaller result, which is not the case for our algorithm. Our method makes more sense for string matching, because one single character change from the legitimate domain name should not be very far away from the original domain names, for example, it should be similar if you change 'hellokitty' to 'hello2kitty'.



(a) HMM detection method 1 for existing DGAs



(b) HMM detection method 2 for new DGAs

Figure 4.11: Two HMM detection methods

With the normalized probability of input domain names calculated from the HMM, we consider two HMM-based DGA detection scenarios:

(1) detecting if the domain names belong to a known DGA;

(2) detecting if the domain names are generated by any DGA, including previously unknown DGAs.

Detection schemes for both scenarios are provided, as illustrated in Figure 4.11. From figure 4.11a, method 1 determines whether the test domain name D is more likely to be generated by the legitimate HMM or botnet DGA HMM by calculating the difference $\Delta P = P_1 - P_2$, where P_1/P_2 is the likelihood that D is generated by the legitimate/DGA HMM. If ΔP exceeds a predefined threshold, D is considered more similar to a malicious domain. This is consistent with how anti-virus companies decide if domain names belong to an existing botnet DGA. Figure 4.11b calculates the probability that D is generated by the HMM inferred from legitimate domain names. It compares D with legitimate domain names only and is suitable for detecting new DGAs, since no samples are available.

4.2.2 Experiment setup & results

To evaluate the HMM-based DGA detection methods, we compared them with three detection methods (KL, ED and JI) [181] as benchmarks. We designed the experiments so that KL/ED/JI are compared fairly with 2 HMM detection methods.

HMM method 1 decides whether or not an input domain name is malicious by comparing it to both legitimate HMM and the botnet DGA HMM. For the purpose of comparison, $\Delta P = P_1 - P_2$ is calculated using KL/ED/JI methods respectively. Note that, we use the same 1000 test datasets for HMM/KL/ED/JI. Each test dataset contains 500 legitimate domain names [4] and 500 malicious domain names from one botnet DGA. For each test domain name D under one detection method, we calculated the corresponding average distance d_l between D and the 500 legitimate domain names, and the average distance d_m between D and the 1000 malicious domain names. Then we compare $\Delta_d = d_l - d_m$ to a predefined threshold to determine if D is benign or malicious. A ROC curve of TPR/FPR pairs is plotted to provide a graphical view of the detection performance of different methods.

The comparison between HMM method 2 and KL/ED/JI detection is carried out in a similar way. HMM method 2 decides whether or not an input domain name is malicious by comparing it to the legitimate HMM alone. We used the same 1000 test datasets as HMM method 1. For each input domain name D under one detection method, we only calculated the corresponding average distance d_l from D to the 500 legitimate domain names, and use d_l to draw ROC curves for different detection methods.

Three tests are performed:

(1) Test HMM method 1 against KL/ED/JI on the *small* DGA dataset, to compare their performance in detecting existing DGAs;

(2) Test HMM method 2 against KL/ED/JI on the *small* DGA dataset, to compare their performance in detecting new DGAs;

(3) Test HMM method 2 against KL/ED/JI on the *large* DGA dataset, which extends test 2) and further verifies that the proposed HMM-based detection works well for detecting new DGAs.

[4] For example, we want to generate 500 third-level sub-domain names under *clemson.edu*. The output domain names will be *ece*, *cs*, etc., because we have *ece.clemson.edu*, *cs.clemson.edu, etc.*.

4.2.2.1 Test 1: HMM detection method 1 on the *small* DGA dataset

In test 1, we intended to use all 12 DGAs in the *small* DGA dataset (Table **??**(a)). However, we were only able to infer 8 HMMs on the 8 DGAs (srizbi, goz, torpig, matsnu, tinba, pushdo, rovnix, and ramdo). The reason why HMM inference fails on the other 4 DGAs (cryptolocker, conficker, new_goz, and zeus) is that their probability distributions are stateless [134, 23]. The probability that any character occurs does not depend on the previous character. This makes them unsuitable for HMM detection method 1. In section 4.2.2.2, we show that all 12 DGAs can be detected using HMM detection method 2.

Figure 4.12 shows the ROC curves of HMM/KL/ED/JI detection on the *small* DGA dataset. We can see that HMM detection outperforms the KL/ED/JI because all points on the ROC curves, regardless of the botnet DGA, achieve almost perfect detection (closest to (0,1)). Among KL/ED/JI detection, JI is slightly better than KL, and ED is the worst.

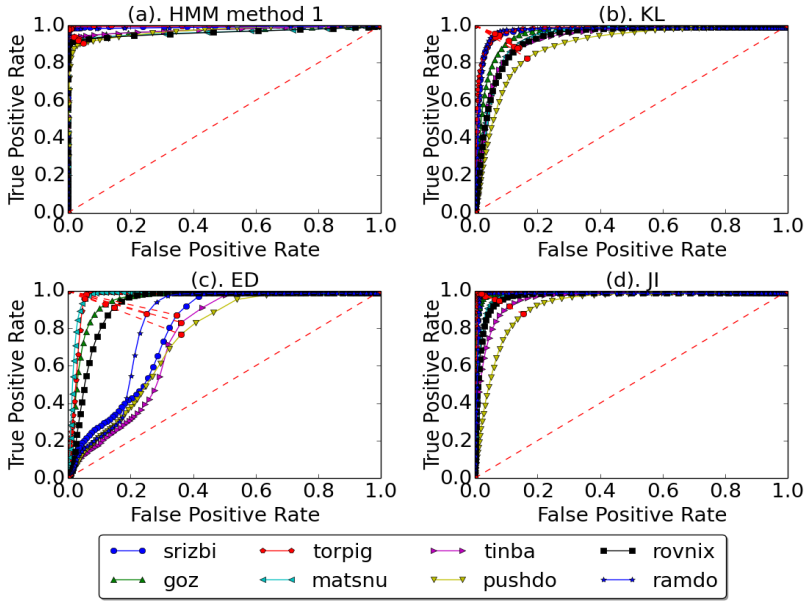


Figure 4.12: ROC curves of (a) HMM method 1, (b) KL, (c) ED, (d) JI detection on the *small* DGA dataset

For different DGAs, the performance order is similar across different detection methods. Note that for all DGAs, HMM-detection is always better than KL/ED/JI. Individually for each detection method, the performance in descending order (from the easiest to be detected to the hardest) is:

- For HMM, torpig > srizbi > ramdo > tinba > goz > matsnu > rovnix > pushdo;
- For KL, ramdo > srizbi > torpig > goz > matsnu > rovnix > tinba > pushdo;
- For ED, torpig > matsnu > goz > rovnix > ramdo > srizbi > tinba > pushdo;
- For JI, torpig > ramdo > srizbi > goz > matsnu > rovnix > tinba > pushdo.

The similarity in the detection performance is that goz, matsnu, rovnix and pushdo always have the worst performance. For some DGAs, like ramdo, detection performance varies depending on the approach. Let us consider three example domain names from ramdo, *aaqmgagwakuwqugu*, *ocmqukucgcqwicqc*, and *mcgwuswqgcosgake*. Although they have the same length of 16 characters and some characters can be repeated, they do not always contain the same letters. Since ED and JI are good at finding similar characters from one domain name to another, they cannot catch the similarity in ramdo domain names. However, they can be caught with KL, because KL works on the letter distribution. The HMM has no trouble catching the deviation between legitimate domain names and ramdo domain names, because the ramdo domain names have different lexical features from legitimate domain names.

4.2.2.2 Test 2: HMM detection method 2 on the *small* DGA dataset

In test 2, we tested the four detection approaches (HMM/KL/ED/JI) on the *small* DGA dataset that contains domains generated using all 12 DGAs. Figure 4.13 shows the ROC curves. Compared to Figure 4.12, we can see that all detection methods are getting worse performance without the comparison against the malicious datasets, which makes sense because we try to label DGA domain names only based on the legitimate domain names.

The detection performance in descendire work to try to cluster the similar domain names before DGAng order is:

- For HMM, goz > ramdo > tinba > cryptolocker > zeus > new_goz > conficker > rovnix > matsnu > srizbi > pushdo > torpig;
- For KL, srizbi > goz > conficker > tinba > ramdo > cryptolocker > zeus > new_goz > pushdo > rovnix > matsnu > torpig;
- For ED, new_goz > zeus > matsnu > goz > rovnix > ramdo > srizbi > pushdo > conficker > tinba > torpig > cryptolocker;

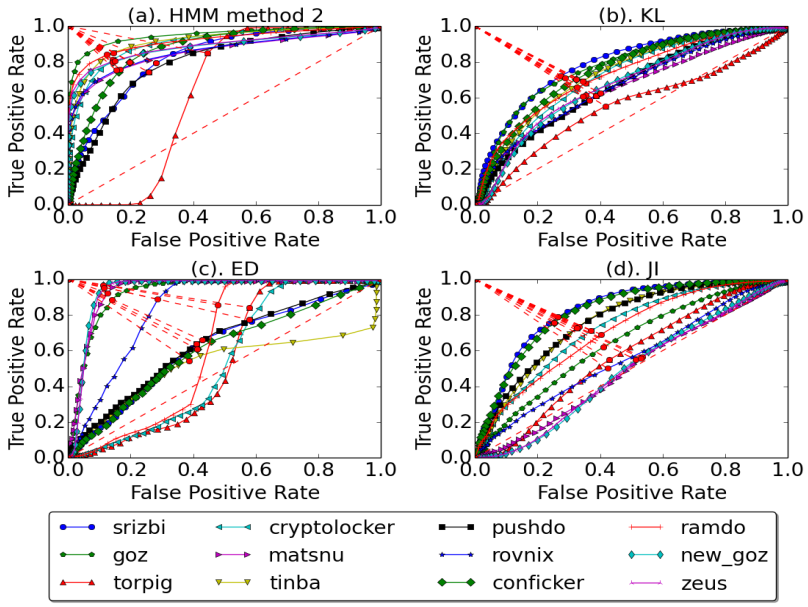


Figure 4.13: ROC curves of (a) HMM method 2, (b) KL, (c) ED, (d) JI detection on the *small* DGA dataset

- For JI, srizbi > conficker > tinba > pushdo > cryptolocker > ramdo > goz > torpig > rovnix > new_goz > zeus > matsnu.

HMM detection method 2 gives a good detection rate with at least 70% TPR and at most 30% FPR on all DGAs except torpig. Torpig has a 85% TPR and 45% FPR, which is unacceptable in practice. Let us look at 3 torpig domain names, *2ah0a1abx3apra*, *7ah3a3ax7apra*, and *10ah0a1abx0apra*. They have the same pattern: four syllables (*ah*, *a*, *ax* and *apra*) with numbers in between them. Such similarity results in little change in discrepancy between different torpig domain names and the legitimate domain names. By contrast, legitimate domains are much more varied. Therefore, it is impossible to find a threshold that can effectively differentiate between legitimate domains from torpig domain names.

Most DGAs can easily evade KL/ED/JI detection. Specifically, for KL on 12 DGAs, all FPRs are around 40% and the TPRs are around 60%, which cannot be used for detection in the real life, because there is almost half and half chance of correct detection and false detection on any DGA. For ED, the detection performance is varied, where the best can be 96% TPR and 11% FPR for detecting new_goz, but the worst can be 77% TPR and 58% FPR for detecting cryptolocker. The detection performance of JI is slightly worse than KL, both of which are not suitable for detecting

new DGAs.

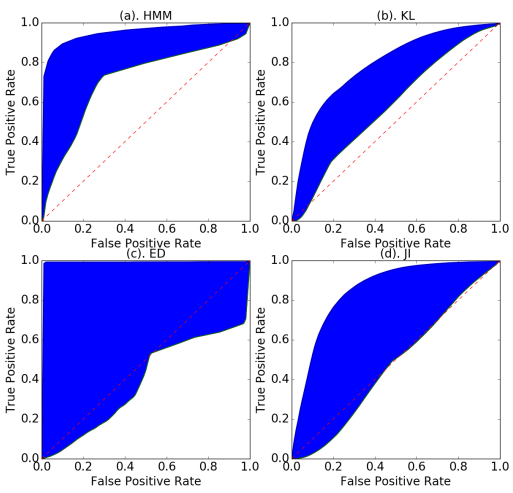
In terms of DGAs, the performance of the same DGA varies under different detection methods. DGAs like torpig are always more difficult to be caught than others irrespective of the detection approach. Some DGAs, on the contrary, are always easier to identify, such as srizbi and conficker. Some DGAs, like matsnu and new_goz, have almost the opposite performance under different detection methods. Let us look at 3 matsnu domain names, *winnerdroprepairwearbone*, *candlesellchairtowerhide*, and *guaranteeraisebeseasontrade*, which are concatenated English words. The reason why ED can catch it is that, since English words have lexical features (for example, vowels appear more frequently than consonants), it will take different steps to transform one to another, compared with legitimate domain names. For KL and JI, especially JI, since there are few shared letters among domain names, KL and JI don't work very well. Since HMM represents the lexical features of legitimate domain names, it is easier to tell the difference when it compares a DGA domain name.

4.2.2.3 Test 3: HMM detection method 2 on the *large* DGA dataset

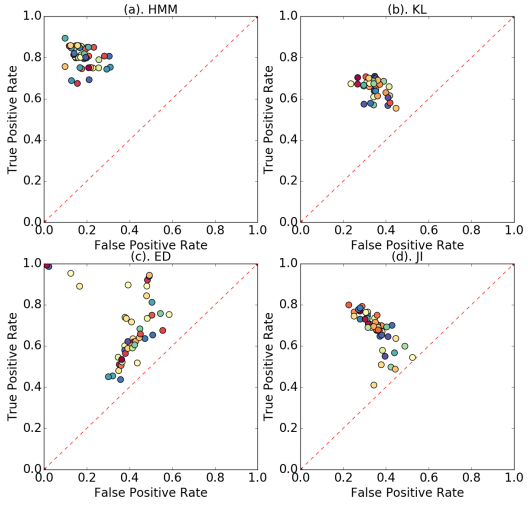
In test 3, we tested HMM detection method 2 on 46 DGAs, which is a larger dataset than the one used in test 2 (12 DGAs). Since it is hard to visualize 46 ROC curves in one figure, we plotted the ROC envelope (the upper and lower bound of all ROC curves) in Figure 4.14a and the best operating points in Figure 4.14b.

In Figure 4.14a, we can see that the envelope for HMM detection method 2 is the smallest and closest to (0,1), which suggests all DGAs are successfully detected by HMM. The envelope for KL is small, but far from (0,1), which suggests that all DGAs are consistently not detected by KL. For ED, the envelope is huge, which takes more than half of the ROC space. It indicates that ED is not stable, and it may give pretty bad detection sometimes. The result of JI is similar to KL, but slightly worse (the points are more scattered in the ROC curve).

In Figure 4.14b, we can see the clusters of the best operating points for different detection methods. For HMM, all best operating points are close to (0,1) and have good TPRs and FPRs. The best detection DGA is However, for KL/JI, the detection is bad, because almost all best operating points are close to $y = x$, which gives the same FPR and TPR. The points of ED are more scattered. Although a few points are very close to (0,1), the performance is not stable.



(a) ROC envelopes



(b) ROC best operating points

Figure 4.14: Comparison among (a) HMM method 2, (b) KL, (c) ED, (d) JI detection on the *large* DGA dataset

4.2.3 Discussion and future work

In Section 4.2.2, we compared the detection performance of the proposed two HMM detection methods with KL/ED/JI detection using two datasets. In this section, we use game theory to further analyze the performance of the HMM detection method 2 with 46 DGAs.

We modelled the game between the botmaster (the one who uses DGAs to evade detection) and security personnel (the one who detects DGAs) as a Two-Person Zero-Sum (TPZS) game [168], because they have conflicting interests. If we can find the DGAs (or detection methods) that the botmaster (or the security personnel) tends to choose for the optimal interest, we can have a better understanding of the arm race between the two parties [49].

We define the pay-off as the distance from the best operating point to the the perfect detection (0,1), which is shown in Table 4.11. The botmaster tries to maximize the distance to evade detection, while the security personnel tries to minimize the distance to achieve better TPR and FPR in botnet detection. The original table is simplified [168] as Table 4.10 by removing the dominated strategies. For example, for the security personnel, there are 4 columns of detection methods as HMM/KL/ED/JI, we compare any two columns one time. If we compare HMM and KL, HMM dominates KL because the pay-off of using HMM is always smaller than KL in detecting all 46 DGAs. Since the security personnel is a minimizer who tries to reduce the pay-off, there is no

Table 4.10: Simplified TPZS between the botmaster and security personnel on 46 DGAs

		Player I: security personnel (minimizer)	
Player II:		HMM	JI
Botmaster (maximizer)	simba	0.157	0.131
	symmi	0.152	0.263

incentive for him to choose KL. This means we can remove the column of KL without affecting the analysis results.

The optimal strategy for the botmaster is to choose simba DGA with probability p, where

$$p = \frac{0.263 - 0.152}{(0.157 + 0.263) - (0.152 + 0.131)} = 0.81 \tag{4.6}$$

The optimal strategy for the security personnel is to choose HMM detection with probability q, where

$$q = \frac{0.263 - 0.131}{(0.157 + 0.263) - (0.152 + 0.131)} = 0.96 \tag{4.7}$$

And the expected value of the game is

$$V = \frac{0.157 \times 0.263 - 0.152 \times 0.131}{(0.157 + 0.263) - (0.152 + 0.131)} = 0.16 \tag{4.8}$$

where V is the expected distance between the ROC curve and point (0,1), when both players stick to their optimal strategy.

For future work, there are several possible directions: (1) try different parameters (dictionary [5] or L) in HMM inference to improve the current HMM detection methods; (2) test HMM detection against other string metrics; or (3) combine other detection features to improve the HMM detection, etc.

4.2.4 Summary

This work proposed 2 HMM-based detection methods and compares them with 3 other detection methods (KL/ED/JI) on 2 DGA datasets. The experiment results show that HMM-based methods outperform the KL/ED/JI detection in detecting both existing DGAs and previously

[5] A dictionary is a training set used to build the HMM. In our work, dictionary refers to a word list containing legitimated domain names. Check the paper [51] for details.

Table 4.11: The arm race between the botmaster and security personnel

		Player I: security personnel (minimizer)				
		HMM	KL	ED	JI	min
	fobber	0.051	0.213	0.356	0.205	0.051
	pushdotid	0.079	0.213	0.306	0.154	0.079
	pykspa2s	0.128	0.248	0.392	0.190	0.128
	cryptolocker	0.063	0.228	0.404	0228	0.063
	ranbyus	0.052	0.225	0.304	0.265	0.052
	simba	0.157	0.216	0.321	0.131	0.131
	qakbot	0.043	0.204	0.327	0.231	0.043
	dyre	0.114	0.270	0.0004	0.384	0.0004
	urlzone	0.066	0.251	0.401	0.235	0.066
	ramnit	0.063	0.224	0.414	0.220	0.063
	torpig	0.077	0.234	0.316	0.162	0.077
	sutra	0.063	0.180	0.241	0.233	0.063
	pykspa2	0.131	0.250	0.365	0.187	0.131
	xxhex	0.069	0.270	0.302	0.191	0.069
	bamital	0.130	0.302	0.0007	0.360	0.0007
	murofet	0.043	0.205	0.291	0.206	0.043
	gameover-p2p	0.021	0.160	0.040	0.294	0.021
	dsnchanger	0.065	0.214	0.339	0.165	0.065
	tinba	0.046	0.200	0.375	0.170	0.046
	symmi	0.152	0.323	0.393	0.263	0.152
	matsnu	0.139	0.400	0.318	0.432	0.139
player II:	padcrypt	0.096	0.255	0.246	0.274	0.096
botmaster	suppobox	0.115	0.355	0.424	0.325	0.115
(maximizer)	pushdo	0.109	0.297	0.320	0.204	0.109
	pykspa	0.074	0.210	0.327	0.150	0.074
	shifu	0.081	0.202	0.302	0.119	0.081
	gozi	0.106	0.352	0.212	0.460	0.106
	emotet	0.051	0.222	0.246	0.255	0.051
	rovnix	0.042	0.247	0.166	0.332	0.042
	proslikefan	0.077	0.192	0.336	0.124	0.077
	blackhole	0.035	0.210	0.243	0.236	0.035
	conficker	0.080	0.198	0.328	0.123	0.080
	virut	0.117	0.180	0.320	0.091	0.091
	oderoor	0.065	0.206	0.336	0.151	0.065
	necurs	0.060	0.213	0.379	0.228	0.060
	corebot	0.066	0.280	0.219	0.400	0.066
	gameover	0.054	0.285	0.018	0.483	0.018
	qadars	0.069	0.289	0.349	0.258	0.069
	srizbi	0.110	0.199	0.297	0.128	0.110
	sisron	0.044	0.196	0.362	0.192	0.044
	nymaim	0.077	0.200	0.324	0.127	0.077
	ramdo	0.040	0.220	0.255	0.242	0.040
	murofetweekly	0.035	0.163	0.0002	0.466	0.035
	banjori	0.090	0.323	0.248	0.381	0.090
	locky	0.069	0.227	0.446	0.196	0.069
	bedep	0.045	0.200	0.355	0.211	0.045
	max	0.157	0.400	0.446	0.483	

unknown DGAs. Particularly, the game theory analysis between all 46 DGAs and different detection approaches is presented. The results further confirms the effectiveness of HMM-based detection methods. The optimal strategy for the security personnel is to use HMM detection with 0.96 probability, and the optimal strategy for the botmaster is to use simba DGA with 0.81 probability.

To our best knowledge, this is the first time that an HMM is used for DGA detection. The results are promising. It is helpful for almost any DGA detection system using lexical features. We hope this paper can help security personnel improve the current detection techniques, and take down botnets in a larger scale.

Chapter 5

Applying botnet technologies to counteract network analysis

In this chapter, we will talk about how to apply botnet technologies to counteract network analysis by providing two applications. Section 5.1 proposed a distributed proxy system using botnet fast-flux to evade censorship and surveillance in West Africa. Section 5.2 designed a covert transport protocol using botnet DGA to counteract network analysis. To our best knowledge, there are the first applications that use botnet technologies to design secure application for normal Internet users.

5.1 Applying fast-flux to counter censorship and surveillance

5.1.1 Background

Freedom of expression is guaranteed by Article 19 of the United Nations Declaration of Human rights. However, network censorship and surveillance have been deployed to suppress the freedom of speech in some countries. According to the Freedom House report on press freedom [68], global press freedom is at its lowest point in 12 years in 2015. Figure 5.1 shows the map of press freedom [68], where only 13 percent of the world's population enjoys a free press and 41 percent has a partly free press.

In contrast to traditional media, it is inexpensive to post news online, especially in poor regions. As more voices become available to the population, fearing a loss of control, repressive

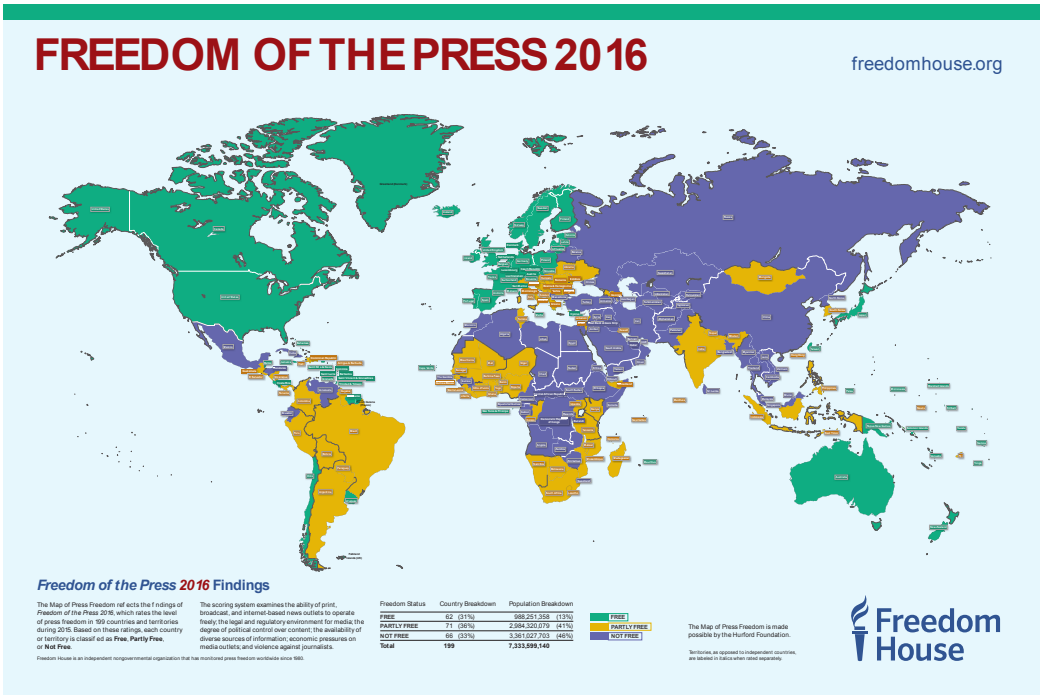


Figure 5.1: Map of freedom of the press in 2016 [68]

governments invest in technologies for surveilling and censoring Internet traffic. National firewalls are built to block or redirect requests to the offending websites. Techniques, including DNS/IP filtering, Deep Packet Inspection (DPI), DNS poisoning etc, are widely deployed in some countries. Since techniques, such as DPI, can also be used in legitimate network management, it is difficult to regulate the export of these technologies.

A number of tools for circumventing censorship exist, notably Tor [120] and Psiphon [73]. Tor and Psiphon both provide proxy services for Internet users. Local software creates a tunnelled network connection to a remote computer. The remote computer executes actions requested by the local host and returns results to the local host through the tunnel. The Tor connection goes through two additional network connections to increase anonymity, while adding some latency and jitter.

Proxy networks, and Virtual Private Networks (VPNs), help users circumvent surveillance and censorship, but are not perfect solutions. They have drawbacks:

(1) A national firewall can track remote connections, detect DNS/IP addresses used by proxies, and block suspect addresses. Censors use Deep Packet Inspection (DPI) to identify addresses with suspect content [174].

(2) Proxy connections can be security risks at both ends. Clients can have sessions spied on by the proxy server. Servers can be made responsible for client actions that seem to be from the local machine.

(3) Increased latency and jitter is a major issue with user acceptance [[125]. Even clients that are aware of censor- ship and surveillance, tend to use faster direct Internet connections.

Tor counters IP address blocking by maintaining a reserve set of previously undisclosed bridge nodes that are deployed gradually. This helps counter IP address blocking. It does not appear to be a long term solution, since the pool of bridge nodes can be depleted sometimes. Our technical goal was to adapt tools widely used in the botnet community to avoiding DNS and IP address filtering. Many botnets remain active for years, despite our best attempts to stop them.

In response to censorship and surveillance, we proposed the idea to developing a distributed proxy system using botnet fast-flux technology. The proxy system used fast flux to change the mapping between IP addresses and DNS names at the same time thus avoiding being detected. It also provided proxy servers in other countries.

5.1.2 Related work

There are some existing solutions that can be used to evade censorship and surveillance. The most famous ones are Tor [120], Psiphon [73], UProxy [75] and Lantern [21].

Tor [120] proxy network tunnels connections through three, separately encrypted, hops. To protect user privacy, entry to the Tor network is normally through a small number of trusted guard nodes. Tor provides advice to help exit node providers minimize their legal risk; primarily by telling ISPs in advance that the node is a Tor exit node. Exit nodes have spied on users in the past. It is unwise to send personally identifiable information (PII) through Tor. Tor's use of two additional network connections to increase anonymity adds more latency and jitter than one hop proxies. Many countries Tor. Iran has blocked SSL/TLS connections, which blocks Tor. China blocks connections to IP addresses that run Tor. China looks for the TLS cipher lists that indicate Tor use. Some countries actively probe and black-list nodes they suspect of providing access to Tor. Tor counters blocking by maintaining a set of reserve (bridge) addresses that become available as needed. Sometimes this set of nodes becomes scarce. Tor is also implementing pluggable transport (PT) layers that modify the network transport layer and disguise Tor traffic. Unfortunately, each

PT is usually supported by only a small number of bridges and a PT can also produce a fingerprint that can be detected.

Psiphon [73] is a one hop proxy. It has multiple modes, including one that hides its use of encryption. Psiphon avoids nation-state firewall blocking by (1) running a large, international park of proxy nodes that are difficult to enumerate, and (2) having access options that obfuscate the connection. The proxy nodes are provided by Psiphon, making Psiphon potentially liable for criminal abuse. Psiphon users have to trust Psiphon not to spy on proxy connections and exploit session information.

UProxy [75] is a browser extension for Chrome and Firefox, which allows users to share their Internet connection with others. It was developed by Google Ideas, but is now the Jigsaw subsidiary of Alphabet, in conjunction with Lantern and the University of Washington. The functionality provided by UProxy is roughly similar to CGI-Proxy, which we used in our system (see Section 5.1.3). Ideally, a friend of a user in a repressive country could volunteer to provide their friend with a proxy connection. In this scenario, the friend risks being potentially responsible for illegal activities done by the proxy client and the client could be spied on by their friend. Alternatively, the proxy connection can be through a commercial provider. This second scenario is basically equivalent to using a commercial VPN. As with Psiphon, the user has to trust the commercial VPN. Many users do not have friends available in countries outside the firewall and countries with national firewalls often block the service providers (Github, GMail, and Facebook) UProxy relies on.

Lantern [21] is a product of the Brave New Software non-profit. It provides a distributed proxy. Lantern bootstraps initial connections through Google Talk servers. Lantern does not provide anonymity, its goal is to provide efficient access to web sites. If the site is not locally blocked, then Lantern will load the material directly and not use the proxy. If the web page is locally blocked, Lantern will retrieve the web page through a proxy connection. Lantern maintains a distributed set of proxies for the user. The user can allow their connection to be shared. All traffic passing through the Lantern peer-to-peer system is encrypted. The distributed nature of Lantern reduces, but does not eliminate, the risks of proxy use. Since only one part of the session would be sent through an individual proxy exit node, the likelihood that an exit node would be blamed for the acts of a malicious user are reduced. Similarly, the amount of information an exit node could harvest from a naive user is reduced.

5.1.3 Distributed proxy system design

5.1.3.1 System summary

Figure 5.2 shows the proposed structure and Figure 5.3 shows the mapped botnet structure. We developed and deployed a network of peer-to-peer proxies for our user community. Our intended users were journalist, human rights activists and political dissidents from the region. Unlike Tor and Psiphon that are open to the public, our tool was meant to be deployed by a small trusted and authorized user community. We vetted the individuals that were invited to our training sessions. These participants were involved in defining the rules that we would enforce in maintaining the network.

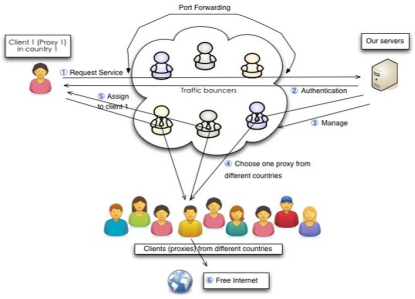


Figure 5.2: Distributed proxy system structure

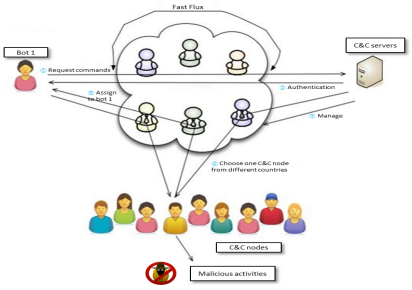


Figure 5.3: Mapped botnet structure

To access the network, the client uses the network protocol in Figure 5.4 to find the address of a remote proxy. The client has a dynamically updated list of DNS names used to connect to our proxy network. It attempts to open an ssh session through a DNS tunnel to our authentication node. Password-less ssh credentials verify that the connection is from one of our authenticated users. When authentication succeeds, the local node receives the DNS name of a proxy clearing-house node.

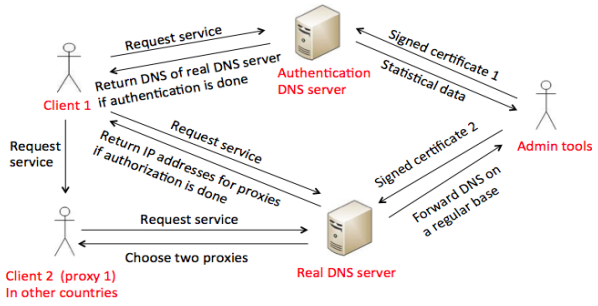


Figure 5.4: Nodes find their remote proxy partner using DNS tunneling to access a proxy clearing house hidden by a fast-flux connection.

The local node opens a second DNS tunnel to the clearing-house and uses scp through the DNS tunnel to retrieve the IP address of the node/proxy that it can use to access the Internet for the current session. The design and implementation include a number of innovations adapted from the world of criminal botnets to counter Internet censorship, especially DNS/IP filtering. The following sections describe the techniques we used to avoid tracking and detection.

5.1.3.2 DNS tunneling

To establish secure communication to our systems authentication servers, we needed to bypass firewalls or network filters. We found that many malicious botnets use DNS to communicate covertly [33].

The domain name system is of interest for many reasons: (1) The domain name service is globally deployed and used; (2) DNS filtering typically blocks attempts to connect to a blacklisted set of sites; and (3) DNS packets and records are rarely validated by the ISP, allowing DNS servers to be impersonated. These factors make the domain name service suited for use as a covert communications channel.

5.1.3.3 Fast flux

Besides DNS tunneling, we also adopted the *fast flux* ideas created by botnets to protect our users. The term fast-flux refers to frequent redefinition of the IP addresses affiliated with a DNS name. In current botnets, one symbolic DNS name will be affiliated with a large number of IP addresses. The IP addresses are given short time to live (TTL) values and swapped out frequently ($<$ 3 mins). The result is a DNS name that can not be reliably tied to any computer through its IP address, see Figure 2.3.

Nodes in the fast-flux tend to work as proxies for a 'mother ship' that wants to be hidden. They effectively avoid detection and tracking by providing a moving target. This is largely why botnets have been so difficult to stop, even when law enforcement and many tech vendors conspire to track them down and neutralize them [146]. Our approach applied this concept to our authentication server and real server to add an additional layer of redundancy and survivability. As with botnet fast-flux, our authentication and proxy connection servers moved frequently to different physical and logical locations on the GENI network [1].

[1]The Global Environment for Network Innovations. `https://www.geni.net/#`

5.1.3.4 Domain name modification

We regularly changed our server's domain names. In practice, we chose our domain names as random words from random Latin alphabet languages taken from Wikipedia. One alternative to this approach would be to algorithmically generate domain names [60] using an algorithm like the one in [52]. To further obscure our network, we used dynamic DNS services to register our domain names. Dynamic DNS services allow individuals to register, at no cost, sub-domains to any of a large set of volunteer root domains. This is useful, since the root domains are quite varied and have no direct connection to our project.

Criminal botnets have on occasion been sink-holed, when a law enforcement agency is able to anticipate the domain name that will be used and register the domain name. This allows the police to identify and isolate the infected nodes, effectively dismantling the botnet.

We had two primary strategies for avoiding sink-holing:

(1) Each node regularly received, during its session, a list of DNS names the authentication server would use in the future when the current DNS name is no longer available.

(2) We created a Tor hidden service with a user forum. Should users become disconnected from the service, we would provide a script on the forum that would provide the system with the current list of DNS names.

In practice, we had no difficulty with sink-holing and never had to use the second approach.

5.1.3.5 Hardened environment

We gave the users a hardened networking environment [22] that was either:

- A bootable, fully encrypted Linux USB drive (Linux Mint[2]),
- A set of scripts that create and remove a temporary work environment on Windows (encrypted using 7-zip[3]), or
- An Android app.

[2]Linux Mint provides full disk encryption, to counter viruses and keyloggers on the user's laptop `https://www.linuxmint.com/`.

[3]7-Zip is an portable software used to compress or zip files secured with encryption`http://www.7-zip.org/`.

A browser using CGI-proxy[4] to access a remote proxy node is launched in the hardened environment. CGI-proxy connections all use TLS, which limits the ability of DPI to identify suspect communications.

Client hardening was implemented so that even if the software is lost or falls into the wrong hands, the risk of user data breach and disclosure of the proxies is greatly reduced. Strong password is required for extracting user data and using our tool.

5.1.3.6 Peer-to-peer proxies

We maintained a number of proxies on GENI. The proxy nodes that we maintained only includes nodes that are currently active. We kept at least four active proxies for about 50 users. The clearing house keeps up-to-date information on the quality of the network connections to proxy nodes, which lets us do load balancing. As shown in Figure 5.4, our user community also act as proxies for each other.

Each user was given the choice whether or not they wanted to act as a proxy for colleagues. This option had not been anticipated at the beginning, but was requested by users during training. We originally assumed that solidarity in the community would lead the participants to provide secure connections for each other. Many participants explained their worries about how authorities could mis-use information harvested by eavesdropping on proxy sessions using their node.

Internet without Borders developed a matrix that identifies countries with either mutual defense agreements or use the same telecommunications providers. Proxy connections are only made through countries that are not friendly with the local government and with different providers. This was done to protect our user community:

- Proxy connections are encrypted while passing from the client through the network to the proxy, but in clear text leaving the proxy. By forcing connections through a country not aligned with the home country, it becomes functionally impossible for the home country's political authorities to survey the session.

- Should a node become compromised and used to harvest the addresses of our users, the home country authorities would only be able to harvest the IP addresses of users in countries they do not have friendly relations with.

[4]CGI-proxy is a tool that lets nodes without web-servers to act as a proxy for others. It can be found at `https://www.jmarshall.com/tools/cgiproxy/`

The user community also became very tight knit and was often aware when a member was arrested. This would also allow us to remove that user's credentials from the authorization node.

5.1.3.7 Lessons learned

During deployment, we learned many important lessons:

- The Internet in Africa is qualitatively different from the Internet found in Europe and the US. Wired connections are rare and power disruptions common. Most connections use 4G wireless in urban areas.
- Start testing the tool in the local environment as soon as possible. Our first version, which had been tested in Europe and the US had extremely poor quality of service on the African networks.
- Enlist local technologist into the project for testing early in the process. Once we started using colleagues in Abidjan and Abuja to test the system, we were able to find timing errors more quickly.
- Use local technologist for technical support for your other users. Many of our web activists were part of co-working spaces in the region, they made themselves available with short notice for helping journalists needing help.
- Listen to the local participants to learn security problems that they have, which you have not anticipated. We had assumed that all users would be eager to serve as proxies for their colleagues. In fact, many users were hesitant to act as proxies for others. This is reasonable for people living under authoritarian regimes and we were naive to not foresee this.
- Take the lessons learned locally and apply them in different situations, this can provide new insights. We took out wargames from the African training and used it as a class exercise for our college students. The students found ways of evading our surveillance that we had not considered. We added those tools to the next set of training sessions.
- The political situation can be changed for the better by the local population when they have access to information. Participants in our training were parts of groups (rap singers and web activists working together) that managed to bring about regime change, removing entrenched governments from power.

5.1.4 Proxy system comparison

We compare here the proxy systems discussed in this paper. The systems vary on how proxy nodes are chosen:

- Tor and Lantern users rely on public proxy nodes. Public proxy nodes have been used to spy on users.
- Psiphon runs its own proxies. Psiphon has access to incoming and outgoing traffic.
- UProxy forces users to find their own proxy nodes.
- We provided users with a community of professionally vetted colleagues that they meet face to face.

The proxies route traffic differently:

- Psiphon connects users directly with nodes located mainly in Western countries.
- When not using the connection of a friend, UProxy uses cloud connections through Digital Ocean (sites in North America, Europe, Bangalore, and Singapore), Facebook, Github, or Google. These can be hard to access from countries with active censorship, for example many are blocked in China.
- Tor routes are chosen from nodes distributed throughout the world. Users can specify preferred nodes (and therefore countries) for entry and exit.
- Lantern's routing assumes individuals are not targeted. Proxy routes include nodes in the local country.
- We assume our users are targeted. We route traffic to proxy nodes located either at US research universities or in a West African country that is not allied with the local government.

To the best of our knowledge, only our proxy explicitly considers political tensions in choosing how to route proxy traffic.

The way Tor, Psiphon, Lantern and UProxy maintain direct connections to proxy nodes has made them vulnerable to traffic fingerprinting, blacklisting and active probing. Our use of fast-flux is different from these existing tools. By frequently changing the DNS and IP addresses associated with our proxy, it should be more difficult to use these techniques to disable our system.

Traffic fingerprinting would still be possible to identify our use of DNS tunneling, however, our use of DNS tunneling requires very few, small messages. To date this has not been a problem.

5.1.5 Summary

We implemented a distributed proxy system to evade the censorship and surveillance in West Africa. The tool that we provided supports Android, Windows and Linux operating system. The goal of helping journalists and human right advocates to have a secure Internet access is achieved. Unlike other proxy-based solutions, our system uses fast-flux techniques to defend against active probing. To our best knowledge, this is the first time where botnet techniques are applied to application where normal Internet users can benefit.

5.2 Applying DGA to a covert transport protocol

5.2.1 Background

Protocol obfuscation is widely used for evading censorship and surveillance, and hiding criminal activity. Most firewalls use DPI to analyze network packets and filter out sensitive information. But if the source protocol is obfuscated or transformed into a different protocol, detection techniques that worked well with the source protocol will either detect nothing sensitive, or detect something which is far from the real information. When encrypted connections draw attention or are blocked [1], protocol obfuscation is a solution for covert communication. For example, consider a botnet mothership trying to diffuse instructions to its infected zombies, obfuscating encrypted data streams is very useful.

In general, protocol obfuscation can be divided into two categories: (1) protocol mimicry, and (2) protocol tunneling. Protocol mimicry is to make protocol A look like protocol B, by tampering some features (packet syntax and statistical features) of protocol A. Protocol tunneling is to take the contents of packets from protocol A and put them into the payload of protocol B. In this way, protocol B is a carrier and masquerade of protocol A. As Houmansadr [67] pointed out, most protocol mimicry fails to be completely unobservable even without the attacker resorting to correlating multiple network flows or performing sophisticated traffic analysis. He concluded that mimicking the protocol in its entirety, including its reaction to errors, typical traffic and artifacts,

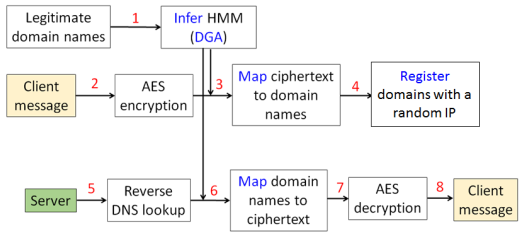


Figure 5.5: Flow chart of the covert data transport protocol

is difficult. Protocol tunnelling, on the other hand, is easy to implement and there are many tools available. However, it is vulnerable to statistical analysis [20, 41].

Inspired by the previous work in DGA, we are thinking of transforming arbitrary network traffic into legitimate DNS traffic [50]. The server encodes the message into a list of domain names and register them to a randomly chosen IP address. The client does a reverse-DNS lookup on the IP address and decodes the domain names to retrieve the message. Different from DNS tunnelling, this doesn't use uncommon record types (TXT records) or carry suspiciously large volume of traffic as DNS payloads. On the contrary, the resulting traffic will be normal DNS lookup/reverse-lookup traffic, which will not attract attention. The data transmission is not vulnerable to DPI.

5.2.2 Protocol design

The covert data transport protocol is based on two-way communications between a client and a server. Figure 5.5 shows that the flow chart of the protocol. Before communication starts, the client and server will share the following information out of band: (1) AES key, (2) pseudo-random number generator (PRG), (3) PRG seed, and (4) an HMM that represents the statistical model of legitimate IPv4 domain names (step 1) and start state.

The HMM guarantees that domain names generated by the HMM will have the same statistical features as, but not conflict with, legitimate domain names. The steps of communication from the client to the server are:

- The client prepares the message ('client message') to send to the server.
- The message is AES encrypted into ciphertext (step 2).
- The client maps ciphertext to domain names with the HMM-encoding algorithm described below (step 3).
- The client registers the generated domain names with the pseudo-randomly chosen IP address

(step 4).

- The server reverse DNS-lookups the chosen IP address and retrieves the domain names (step 5).
- The server maps domain names back to ciphertext (step 6).
- Ciphertext is AES decrypted into client message (step 7 and 8).

5.2.2.1 HMM-encoding/decoding algorithm

HMM-encoding maps data strings (ciphertext in our work) to domain names. Since the HMM is a probabilistic regular grammar with transitions associated with different probability, an intuitive idea is to find a path in HMM on one side, which can be recovered on the other side. This requires both sides to choose a start state and the same encoding/decoding algorithm. We choose the state with the largest asymptotic probability as the starting state because it is the state occurring with the largest probability as time goes to infinity.

To encode the message into the path of HMM, we round transition probabilities to the closest $\frac{1}{2^n}$, where n is an integer. Rounding keeps the statistical features of legitimate domain names, while finding a way to encode and decode. Since the original probabilities of all transitions going out of a state sum up to 1, it is possible to round them to the closest $\frac{1}{2^n}$. The set of transition probabilities associated with state S_i is denoted as $\{P_{i1}, P_{i2}, ..., P_{ik}\}$, where k is the total number of transitions leaving state S_i and $\sum_{n=1}^{k} P_{in} = 1$. The set of rounded transition probabilities associated with state S_i is denoted as $\{RP_{i1}, RP_{i2}, ..., RP_{ik}\}$, where k is the total number of transitions leaving state S_i and $\sum_{n=1}^{k} RP_{in} = 1$.

With the rounded transition probabilities of state S_i, we associate a binary representation for each transition. This allows us to encode binary data (converted from the original 'client message') into the HMM. Algorithm 2 shows the binarization algorithm. The input is the rounded probability ($\{RP_{im}\}$, $m = 1, ..., k$), and the output is the binary representation ($\{BIN\_RP_{im}\}$) for the corresponding transitions. Note that any string can be encoded with the binary representations.

After binarization, each transition of the HMM has a binary representation. Given the binary data, there is a unique path going through the HMM that encodes the binary data string. However, there are two technical difficulties:

- The output symbols in HMM includes 'a-z', '0-9', '-' and '␣' (space). Normally, domain names generated from the HMM are separated with space (for example 'ab␣cd␣ef'. But the receiver

Data: Rounded transition probabilities associated with state S_i is denoted as $\{RP_{im}\}$;
Result: Binarized transition probabilities $\{BIN\_RP_{ij}\}$;
Step 1: Order $\{RP_{im}\}$ from the largest to the smallest into $\{ORP_{i1}, ..., ORP_{ik}\}$;
Step 2: Start with the largest value ORP_{i1}, and $BIN\_ORP_{i1} = \underbrace{0...0}_{\log_2 \frac{1}{ORP_{i1}}}$;
Initiate j = 2, last_input = ORP_{i1}, last_output = BIN_ORP_{i1}, and $num = \log_2 \frac{1}{ORP_{i1}}$
while $j \leq k$ **do**
  if ORP_{ij} == *last_input* **then**
    BIN_ORP_{ij} = last_output + 1 ;
  else
    calculate $diff = \log_2 \frac{1}{ORP_{ij}}$ - num;
    BIN_ORP_{ij} = (last_output + 1)*2^{diff};
    update $num = \log_2 \frac{1}{ORP_{ij}}$;
  end
  update last_input = ORP_{ij};
  update last_output = (last_output + 1)*2^{diff};
  j = j + 1;
end
From the mapping between $\{RP_{ij}\}$ to $\{ORP_{ij}\}$, obtain $\{BIN\_RP_{ij}\}$ from $\{BIN\_ORP_{ij}\}$ and return $\{BIN\_RP_{ij}\}$.

Algorithm 2: Probability binarization algorithm

will not be able to put space in the correct spot (only put space after 'ab' and 'cd', but not 'ef').

- With receiving the domain names ('ab', 'cd' and 'ef'), the receiver doesn't know the order.

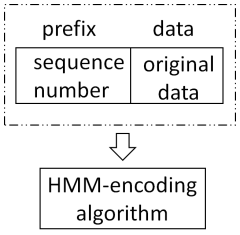


Figure 5.6: The new data structure

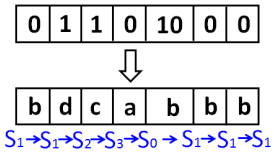
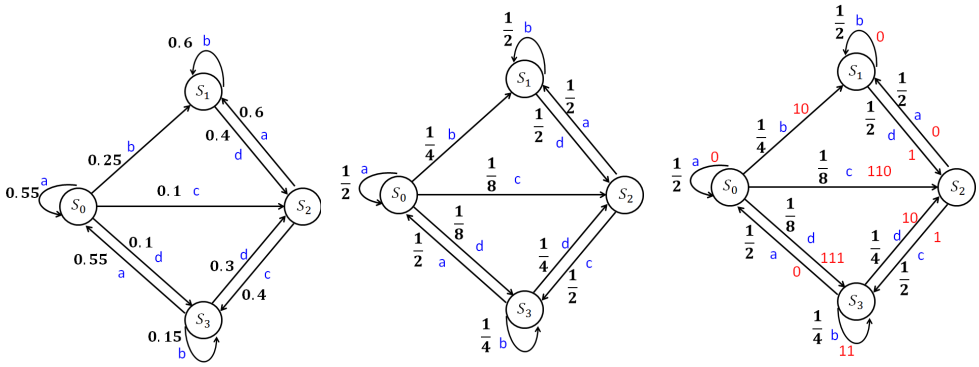


Figure 5.7: An example mapping

The solution is to add the prefix to the original data. The new data structure is in Figure 5.6. The prefix is the sequence number, which guarantees the correct order of the message on the receiver side.

After the client message is converted into multiple domain names, we register them as sub-domain names with the server IP address using dynamic DNS service (https://freedns.afraid.org/ for example).



(a) An example HMM (b) HMM after probability rounding algorithm (c) HMM after binarization algorithm

Figure 5.8: An illustrative example

To decode the domain names into client message, the server will reverse-lookup a randomly chosen IP address and retrieve a list of sub-domain names. After doing the inverse of the encoding, several message pieces are recovered. Using the sequence number, the original client message is obtained.

Since both client and server will register the same randomly chosen IP address to make the system work, we have to consider:

- The probability of choosing a collided domain name is almost zero if using IPv6 domain names. The IPv6 address space range contains 2^{128} entries, which is more than one entry for every three atoms in the universe. If an address is chosen at random, the chance of a collision is essentially zero. If the entire IPv4 space were to choose IPv6 addresses at random, the probability of at least one collision is approximately 0.0155.
- If collision happens, the system will work fine. Because the HMM decoding won't work on the existing domain names. By simply ignoring the domain names that fail to pass HMM decoding, we can separate the desired domain names from the existing ones.

Besides the two-way communication, the idea can also be used in botnets where the botnet C&C encodes the message into domain names, and the bot retrieves the message. This will make the botnet traffic look like pure DNS traffic.

Table 5.1: Performance of HMM encoding/decoding

Input String Length	AESed String Length	Encode Execution Time	Decode Execution Time	Output Domain Names
11	24	267.98s	114.04s	lviazea01; lviahgeurakk-04; lvia79djqb02; lviamhlayae2-; lviabpi62f03; lvielm13owul; lvienh103; lvwlei14j0volg002; lvwltrti102
22	44	460.03s	188.05s	lviaudzsudzssic20e0002; lvwlynaks; lvia523rajc10j02; lvia76rptu16q603; lviabpm01; lvwleg186bczp; lvienb-8702; lvwlejo01; lvwlcko1-02; lvwlt-28d6gyopur24-05; lvwl07bsjm-1vusfoe-1ofw23ucm29c58-05;
34	64	781.06s	191.32s	lviauket99jd104; lviaha102; lvia78h; lviamiltathybpr5102; lvienbpo-04; lvwlep003; lvwltb5eud01; lvwl1d1601; lvwlyunt9103; lvwlxmanrovp; lvwlcl67ds01; lvw6l0v30ge-nsigoyed6gy-01; luctgold99q; luctgjnbu9jv5wk; luctiwaa; luctigh5; luctia-thne55x001; luctin01; luctivbusovalriq5ilf2l5lld-z2tchl0lcl2201

5.2.2.2 An illustrative example

The HMM representing the real legitimate domain names is very complex (it has 2995 states and each state has at most 38 outgoing transitions). We use an example HMM (Figure 5.8a) to illustrate the algorithm. In the example HMM, the asymptotic probability of state S_i $(i = 0, 1, 2, 3)$ is calculated as: $P(S_0) = 0.15$, $P(S_1) = 0.46$, $P(S_2) = 0.25$ and $P(S_3) = 0.14$. So S_1 is chosen as the starting state for both the client and the server.

After the probability rounding algorithm, all transition probabilities are rounded to the closest $\frac{1}{2^n}$ (Figure 5.8b). After the binarization algorithm, each transition is associated with a binary representation (Figure 5.8c).

Suppose that the 'client message' is 'h', its binary representation is '01101000'. The starting state is S_1. Following Figure 5.8c, the state transition is $S_1 \to S_1 \to S_2 \to S_3 \to S_0 \to S_1 \to S_1 \to S_1$. The mapping is in Figure 5.7. With the HMM-encoding algorithm, the input 'h' is converted into 'bdcabbb'.

5.2.3 Performance

We implemented the proof-of-concept idea on a laptop (OS: Windows 10, CPU: i5-5300U, RAM: 8G). We measure the encoding and decoding execution time on various lengths of input string.

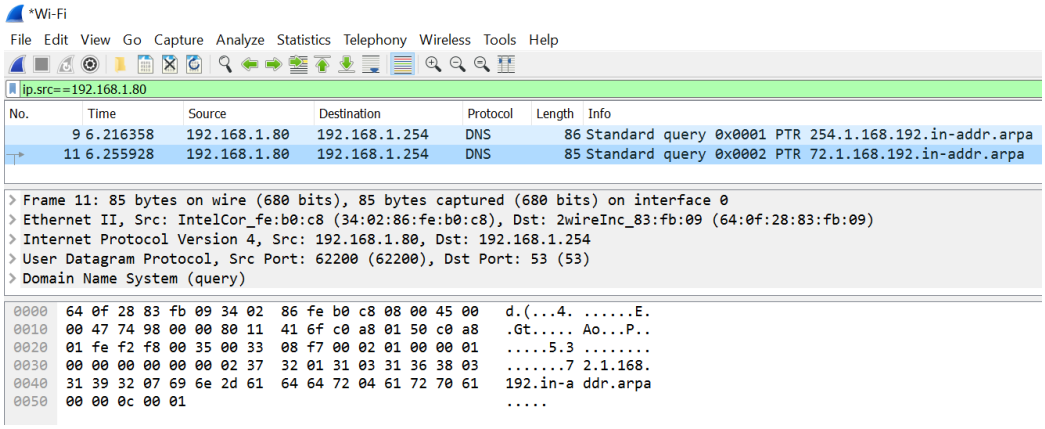


Figure 5.9: Wireshark screenshot

Table 5.1 shows the performance of the transparent protocol. For AES encryption/decryption, we use block size equal to 16. This will make the length of AESed-string 24 (when length of the input string is between 1-15), 44 (when length is between 16-31), 64 (when length is between 32 to 47) bytes etc. So the performance of the implementation is more dependent on the length of AESed-string than the length of the input string. Note that the input string can be any length.

From Table 5.1, we can see that the execution time is proportional with the length of AESed-string. Note that, all output domain names start with 'l', which is resulted from the same start state (the state with the largest asymptotic probability). How to pick different start states where the client and server agree on will be an interesting topic as the future work.

Figure 5.9 shows the network traffic sniffed by wireshark. The server registers the domain names with a randomly chosen IP address (192.168.1.72). The client (192.168.1.80) is trying to retrieve a message from the server. All communication happens between the client to the local DNS server (192.168.1.254). And the DNS server cannot distinguish the DNS traffic from the others. This proves the destination IP address stays hidden and the idea works!

5.2.4 Security Analysis

To further validate the possibility of the implementation, we conduct security empirical analysis in terms of confidentiality, differentiability and communication.

- Confidentiality: Since the arbitrary network traffic is masquerading as normal DNS traffic, it is hard to filter it out from other DNS traffic. Even if a man-in-the-middle (MITM) filters out

the traffic and tried to decode the DNS packets, he/she has to have all pre-shared information: AES key, PRG, PRG seed, and the HMM. This is infeasible unless he has the software package.

- Differentiability: Except for normal DNS queries, the application involves reverse-DNS lookup and DNS registration traffic. Reverse-DNS lookup is widely used by common security tools [173] including network troubleshooting tools, anti-spam techniques, and system monitoring tools. For example, email anti-spam software checks the domain names using reverse-DNS lookup to see if the source is a dynamically assigned address, which is unlikely used by a legitimate mail server. Web browsers use reverse-DNS lookup to verify the same origin of the requests to avoid DNS rebinding attacks [129]. In terms of DNS registration traffic, although large companies like Google and Amazon keep registering domain names in a round-robin fashion, it remains unclear that how unusual the DNS registration traffic is. This will be an interesting topic for future work.
- Communication: If there is an ISP-level surveillance tool looking for this application, it is impossible to pick up the related domain names. Even if it is possible, we can use Tor at the client side (who generates the domain names) to further hide the source IP address. Also, the registered IP address associated with the generated domain names is randomly selected, so there is no way to trace the communication parties using the IP address.

5.2.5 Summary and Future Work

This paper proposes the structure of a new covert data transport protocol. It would be suitable for botnet C&C. The advantages over existing protocol obfuscation tools are that network traffic is real and normal DNS lookup traffic on benign-looking domain names, which is protected against DPI detection. One possible disadvantage is the low throughput, which is more suitable for delay-tolerant communication than real-time chatting, such as twitter-like social networks, sensitive information retrieval tools etc. We hope this paper will provide a new insight into protocol obfuscation.

Future work can tune parameters to optimize the throughput including re-designing HMM encoding/decoding algorithm, better data structure to improve the performance etc.

The proposed concept will be helpful in moving botnet countermeasures to being pro-active. It will change the routines of botnets innovating, anti-virus vendors reverse-engineering and finding countermeasures [49]. Instead, we need to develop better technology than the enemy and know how

to counter botnets before they deploy innovations. Up to now, the botnet implementers have the advantage of creating the innovations. By designing countermeasures to better botnet designs in advance, it would make botnet deployment less profitable.

Chapter 6

Protocol obfuscation techniques to counteract network analysis

Botnets use protocol obfuscation techniques to hide their traces. In this chapter, we designed a protocol obfuscation technique to transform an arbitrary sensitive protocol to a benign-looking protocol in terms of syntax and side-channel features. We provided two applications to illustrate the design. Section 6.2 discussed the design of protocol obfuscation technique. Section 6.3 provided application 1 where botnet traffic (Zeus) is transformed to smart grid synchrophasor protocol (TCP to TCP offline transformation). Section 6.4 provided application 2 where Tor traffic is transformed to Network Time Protocol (NTP) and a video game protocol (Minecraft), which was an example of online transforming HTTP to UDP/TCP.

6.1 Introduction

Protocol obfuscation is widely used for evading censorship and surveillance, and hiding criminal activity. Most firewalls use DPI to analyze network packets and filter out sensitive information. But if the source protocol is obfuscated or transformed into a different protocol, detection techniques that worked well with the source protocol will either detect nothing sensitive, or detect something which is far from the real information. When encrypted connections draw attention or are blocked [1], protocol obfuscation is a solution for covert communication.

Also, malware uses traffic camouflaging to hide network traffic. Spyware Taidoor, a banking trojan discovered in 2008, used a Yahoo blog as a botnet C&C server [18], the malicious traffic was camouflaged as HTTP requests to blogs. According to Kaspersky Lab [152], ChewBacca and evolved Zeus use the Tor network to hide communications. Pushdo malware [118] included domain names of large companies or famous educational institutions in the C&C domain list, which made network traffic analysis more difficult. The Morto Trojan [128] sent false DNS requests to a DNS server, which was a C&C server, and the exchanged message was obfuscated by a simple Base64 encoding. The proposed method can transform arbitrary traffic to a target protocol, so that malware detection techniques looking for features of the original protocol won't work.

In this chapter, we present two projects that develop protocol obfuscation techniques to counteract network analysis. The first project can transform Zbot C & C traffic to benign Phasor Measurement Unit (PMU) traffic, which can be read correctly by the PMU sensor. The second project can transform arbitrary network traffic to 2 benign network protocols: Network Time Protocol (NTP) and a video game protocol (Minecraft).

Both use similar designs with revised Format-Transforming Encryption (FTE) [36] and side-channel massage (SCM). The idea is to mimic a target protocol in terms of syntax and side-channel features (packet size and inter-packet time delay). Revised FTE handles the syntax and SCM handles the side-channel features. Both of the syntax and side-channel features of the target protocol are inferred from real network traffic and represented as HMMs. The same idea can be applied to transform any arbitrary source protocol to any chosen target protocol. The two projects illustrate the details of protocol identification, protocol choices, HMM inference, and implementation of transforming TCP to TCP (Application 1) and TCP to NTP (Application 2). Note that application 1 is an offline transformer, while application 2 is an online transformer.

6.2 Designs

6.2.1 Revised Format-Transforming Encryption (FTE)

The input of FTE includes the message to encrypt and the regular expression describing the target protocol. The output ciphertext matches the input regular expression. There is a parameter, called $fixed\_slice$, which refers to the number of bytes that FTE receiver-side should decrypt of an incoming stream. For example, the target regular expression is '$\wedge(a|b)+\$$', which matches an

arbitrary long string only comprising a or b. Given a message *helloworld* and $fixed\_slice = 512$, a string of *aaaaaaaababababaab...bababbabaaababab* containing 512 characters is obtained.

The goal is to transform Zbot traffic into a valid PMU data stream. PMU data is collected to infer a regular expression for synchrophasor protocol. Table 6.1 shows the structure of a PMU data packet [72], and the number of observed values for all fields. The extracted regular expression that fully describes observed PMU traffic is very complex. The transformation using this regular expression requires excessive computational resources.

Table 6.1: PMU packet fields in application layer

Byte Poisition	Length in Bytes	Parameter Name	Number of observed values
0 - 1	2	Synchronization word	1
2 - 3	2	Frame size	1
4 - 5	2	PMU/DC ID number	1
6 - 9	4	SOC time stamp (UTC)	3958
10	1	Time quality flag	4
11 - 13	3	Fraction of second (raw)	30
14 - 15	2	Flags	4
16 - 271	8 each	Phasor #1 - #32	118713 each
272 - 275	4	Actual frequency value	52
276 - 279	4	Rate of change of frequency	7573
280 - 281	2	Digital status words	1
282 - 283	2	PMU checksum	54887

In this study, the FTE performance if improved by mapping observed PMU traffic directly to a regular expression. The Zbot traffic is first transformed with FTE using an example regular expression '$\wedge[0-9a-f]+\$$'. The output string consists of a sequence of hexadecimal symbols. There are 16 possibilities for each character position. Each character position corresponds to one PMU packet field. For each PMU packet field (Figure 6.1), the observed values of that field are divided into 16 groups and assign each group a unique symbol. Based on the predefined mapping relation from hexadecimal symbols to groups of value, for each character in the output string of FTE, a value is randomly picked from the related group.

Note that the previous procedure only applies to fields with at least 16 possible values. For fields with less than 16 possible values, a random value from the observation is used. For example, suppose the FTE output is 'a10b5d7...'. The first character 'a' corresponds to group number 10. So it can be replaced by a random value in the 10th group of the first field of PMU packet, which is 'SOC time stamp (UTC)'.

There are several advantages of the proposed mapping technique: (1) The input regular expression is straightforward. There is no need to extract regular expressions from target protocols. (2) Since regular expression extraction is not needed, the process can be easily automated. (3) The proposed mapping uses observed data, so the output is closer to real PMU traffic than traditional FTE. (4) The throughput can be increased by adjusting the number of symbols used to represent one protocol field.

6.2.2 Side-Channel Massage (SCM)

Timing SCM modifies inter-packet delays to match the target protocol to evade side-channel analysis. Given collected inter-packet delays of the target protocol, a deterministic HMM representing the timing pattern is constructed. An HMM is a statistical Markov model in which the modeled system is a Markov process with unobserved (hidden) states. The standard HMM in [122] has two sets of random processes: states and outputs (observations). In a deterministic HMM, there is only a single set of random processes. In this model, the state transition labels are uniquely associated with an output alphabet.

To infer the HMM representing the timing pattern of PMU traffic, packet collection is performed on the PMU side. The time difference, $\triangle t$, is calculated by subtracting the receive time of the previous packet from the time of the current packet. In other words, $\triangle t_i = t_i - t_{i1}$ where t_i is the receive time of packet i. Obviously, the first packet in the data sequence will not have a value, so the process start with $i = 2$. This step is done so that network latencies between the source and the destination are filtered out. Using the calculated values of $\triangle t$, the data is symbolized by grouping it into ranges and assigning anything in that range a unique symbol such as a or b. The symbolization result of is shown as Figure 6.1. Given the symbolized data sequence, the zero knowledge HMM inference algorithm introduced in [134, 184, 24] is used to create the deterministic HMM. The inferred model is shown in Figure 6.2.

Given the model, the counterfeit PMU starts the timing SCM process by randomly selecting a start state in the model. To send the packet, a transition is taken from the current state and the corresponding delay is waited before sending a packet to the PDC. If there are more than one possible transitions out of current state, the transition is chosen randomly, weighed on the probability of each transition.

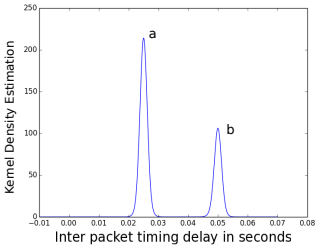


Figure 6.1: The symbolization of inter-packet timing delays of legitimate PMU traffic

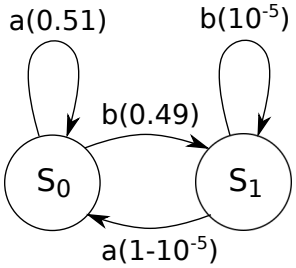


Figure 6.2: The HMM of inter-packet timing delay side-channel of legitimate PMU traffic

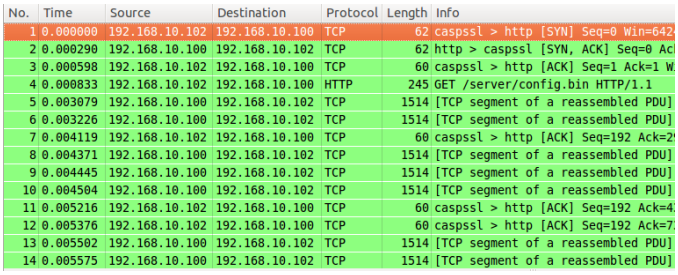
6.3 Application 1: Mimic arbitrary benign protocols

6.3.1 Background

Malware authors conceal Command and Control (C&C) traffic by masquerading as legitimate web browsing activity (HTTP or encrypted HTTPS) [58] or Internet relay chat (IRC). Twitter and DNS queries have also been used to hide C&C communication channels [128]. Detection techniques based on reverse engineering, side-channel analysis, etc., have emerged [17, 92]. Reverse engineering of malware develops network signatures for malicious traffic filtering [148]. Many bots use cryptography or obfuscation to disrupt signature-based detection [7], which can be foiled by blocking all unknown encrypted traffic [104]. However, this has two drawbacks: (1) discouraging the use of cryptography makes traffic less secure and (2) any false positives disrupt legitimate traffic producing denial of service.

Other research uses traffic analysis for detection. Chen [91] presents timing side-channel analysis for Zbot detection using hidden Markov model (HMM) and probabilistic context-free grammar (PCFG). Passive DNS traffic analysis system for detecting and tracking malicious flux networks of a botnet is proposed in [117]. Other traffic side-channels include packet size [193].

As an alternative idea to hide the network protocol, we are thinking of transforming an arbitrary protocol into another. As an example, we are going to camouflage malware traffic as an innocuous application protocol using format-transforming encryption (FTE) [36] and side-channel massage (SCM). FTE converts traffic flow to another application protocol using a regular expression describing the target protocol. This thwarts existing detection approaches because they detect that the infected machines are using IRC, HTTP/HTTPS or DNS to communicate. Additionally, the FTE output is processed using SCM to modify side-channel features, making the system resistant



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.102	192.168.10.100	TCP	62	caspssl > http [SYN] Seq=0 Win=642
2	0.000290	192.168.10.100	192.168.10.102	TCP	62	http > caspssl [SYN, ACK] Seq=0 Ac
3	0.000598	192.168.10.102	192.168.10.100	TCP	60	caspssl > http [ACK] Seq=1 Ack=1 W
4	0.000833	192.168.10.102	192.168.10.100	HTTP	245	GET /server/config.bin HTTP/1.1
5	0.003079	192.168.10.100	192.168.10.102	TCP	1514	[TCP segment of a reassembled PDU]
6	0.003226	192.168.10.100	192.168.10.102	TCP	1514	[TCP segment of a reassembled PDU]
7	0.004119	192.168.10.102	192.168.10.100	TCP	60	caspssl > http [ACK] Seq=192 Ack=2
8	0.004371	192.168.10.100	192.168.10.102	TCP	1514	[TCP segment of a reassembled PDU]
9	0.004445	192.168.10.100	192.168.10.102	TCP	1514	[TCP segment of a reassembled PDU]
10	0.004504	192.168.10.100	192.168.10.102	TCP	1514	[TCP segment of a reassembled PDU]
11	0.005216	192.168.10.102	192.168.10.100	TCP	60	caspssl > http [ACK] Seq=192 Ack=4
12	0.005376	192.168.10.102	192.168.10.100	TCP	60	caspssl > http [ACK] Seq=192 Ack=7
13	0.005502	192.168.10.100	192.168.10.102	TCP	1514	[TCP segment of a reassembled PDU]
14	0.005575	192.168.10.100	192.168.10.102	TCP	1514	[TCP segment of a reassembled PDU]

Figure 6.3: Screenshot of botnet traffic (part)

to side-channel analysis. The motivation of this work is to reveal the weakness of existing traffic detection techniques and show that more sophisticated malware traffic detection solutions are needed.

6.3.2 Traffic collection

A Zeus C&C server and a Zeus bot are set up in the lab on two Windows XP virtual machines. Figure 6.3 shows example botnet traffic. There are two protocols used: HTTP is for communication with the C&C PHP server, and TCP for information exchange between bot and botmaster. The volume of the collected bot traffic is 1.4MB.

In the Real-Time Power and Intelligent Systems (RTPIS) Laboratory of Clemson University [126], a real-time power system is simulated with one hardware PMU and software openPDC (as PDC). Network traffic between PMU and PDC are collected with Wireshark on the same machine (windows 7 OS) running openPDC software. The traffic collection lasts around one hour, and the volume of collected traffic is 108.9MB.

6.3.3 Experiment Setup

The experiment setup is in Figure 6.4. The implementation includes a client-side program (counterfeit PMU program) deployed on the bot and a decoder integrated as a front end of C&C server. The client-side program takes the Zbot data stream as input, processes it with FTE then SCM. It then sends the false PMU data over the network. When the data arrives at the server side, it is decoded by the FTE decoder and forwarded the to C&C server.

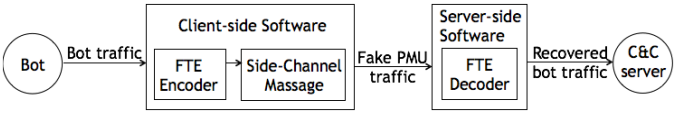
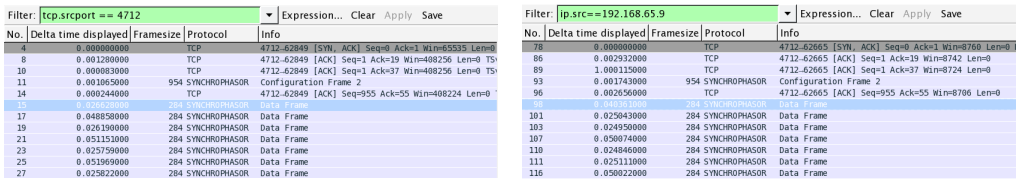


Figure 6.4: Experiment setup.

6.3.4 Results and analysis

It is found that the bot and its C&C server communicate fluently with the deployment of the traffic camouflage system. This indicates the false PMU data is decoded correctly before forwarded to the C&C server.

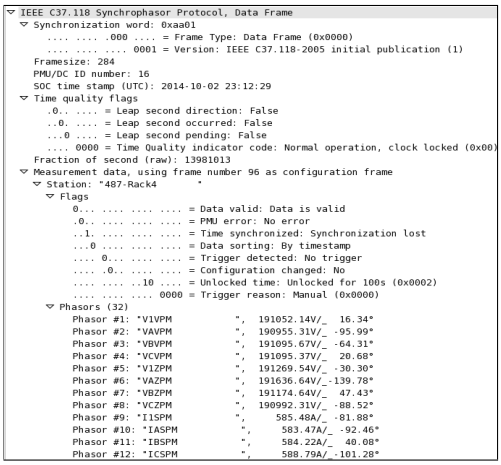
As shown in Figure 6.5a, the false PMU packets are recognized as the synchrophasor protocol, i.e., the protocol used for PMU-PDC communication. The packet details of false PMU displayed in Wireshark are shown in Figure 6.6a, which contain all the fields of a standard PMU packet as shown in Table 6.1. Values of each field are legitimate because they are consistent with the real PMU packets. In particular, the voltage and current measurements are all within a reasonable range compared with real PMU packet shown in Figure6.6b.



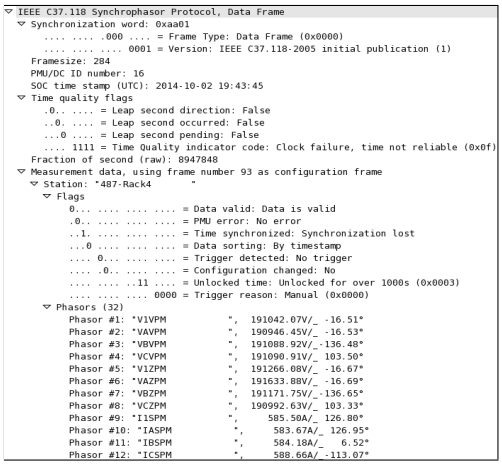
(a) Screen-shot of fake PMU traffic (part) (b) Screen-shot of real PMU traffic (part)

Figure 6.5: Comparison between fake PMU traffic and real PMU to PDC traffic

To further evaluate the proposed approach, side-channel analysis in [92] is launched against the false PMU data flow. The confidence interval (CI) approach in [23] is used to determine if the observed false PMU traffic matches the PMU HMM (see Figure 6.2). The inter-packet delays of the false PMU packets are computed and symbolized. Given the symbolized string, the CI method traces the data back through the HMM. Meanwhile, transition probabilities and corresponding confidence intervals are estimated by frequency counting. This process maps the observation data into the HMM structure. After the confidence intervals are estimated, the percent of transition probabilities from originally given HMM that fall into their respective confidence interval are determined. If this percentage is greater than a threshold value, it is accepted that the observations adequately match the HMM, i.e., the collected PMU traffic is legitimate in terms of timing.



(a) Screen-shot of a typical camouflaged Zbot packet (part)



(b) Screen-shot of a typical PMU measurement packet (part)

Figure 6.6: Comparison between camouflaged Zbot packet and PMU measurement packet

Receiver operating characteristic (ROC) curves are usually used to find the optimal threshold value. By varying the threshold from 0% to 100%, the criteria for acceptance is progressively increased. Since the HMM (see Figure 6.2) in this experiment only contains 2 states and each state has one probability distribution, there are only 3 possible values for the percentage of probability distributions that match, which are 0, 50%, 100%. So there is no need for a ROC curve. The detection results using these three values are given in Table 6.2, where TPR denoted true positive rate, FPR denotes the false positive rate, and l is the threshold used for detection. TPR means the percentage of real PMU traffic that is correctly identified by Wireshark, while FPR means the percentage of false PMU traffic that is identified as PMU traffic. As shown in the results, it is impossible to obtain a high TPR and low FPR at the same time no matter how l varies. Therefore, it is difficult to perform an effective side-channel analysis.

Table 6.2: TPR-FPR with different thresholds

Threshold l	$l = 0$	$0 < l \leq 0.5$	$0.5 < l \leq 1$
TPR	1	1	0.0193
FPR	1	0.9611	0.0352

A communication channel is built between the counterfeit PMU and software OpenPDC. All false PMU traffic is accepted by OpenPDC. This shows that the proposed method is capable of producing false PMU data stream close to real PMU.

6.4 Application 2: Hide arbitrary sensitive protocols

6.4.1 Background

The Onion Router (Tor) [120] provides an infrastructure for anonymous communication over a public network. The idea is to re-route network traffic through several browser-based short-lived proxies (called relay nodes) to evade surveillance and censorship [44]. However, as Tor becomes well-known, some unpublished relay nodes (bridges) are blocked in authoritarian countries [174].

To counter that, obfsproxy [119] is developed to circumvent censorship by camouflaging the Tor traffic between client and bridge. It supports multiple protocols, called pluggable transports, which specify how traffic is transformed. ScrambleSuit [175] is a polymorphic network protocol to obfuscate the transported application data to defend against active probing and protocol fingerprinting. SkypeMorph [105] is a Tor pluggable transport to reshape Tor packets to resemble Skype calls. StegoTorus [171] first uses chopping to change packet sizes and timing information, and then uses steganography to disguise Tor traffic as a message in an innocuous cover protocol, such as HTTP. FTE [36] evades regular-based deep packet inspection (DPI) technologies by transforming Tor traffic into a predefined format. Although FTE is effective in mimicking the contents of packets, it does not mimic timing information, which is vulnerable to side-channel analysis. The proposed SCM method 'massages' the timing side-channel to ameliorate FTE.

This work implemented, tested and fielded *DoppelgaengerProxy*, a new pluggable transport layer for Tor. This transport layer mimicked two target protocols as DoppelgaengerProxy1 (mimic NTP) and DoppelgaengerProxy2 (mimic Minecraft). For each protocol, we performed revised FTE to:

- Hide the contents of the Tor session from local surveillance, and
- Make the session contents consistent with the host protocols.

Our innovation is to integrate protocol side-channel statistical analysis with FTE for target protocols, and then modify the Tor traffic so that the traffic is consistent with the target protocol. System architecture is in Figure 6.7.

Different from application 1, the Tor PT that we presented here is an online transformer. Because of the time limit of the project, we were able to develop a TCP forwarder which can transform arbitrary TCP to NTP/Minecraft. This is a proof-of-concept application before we can

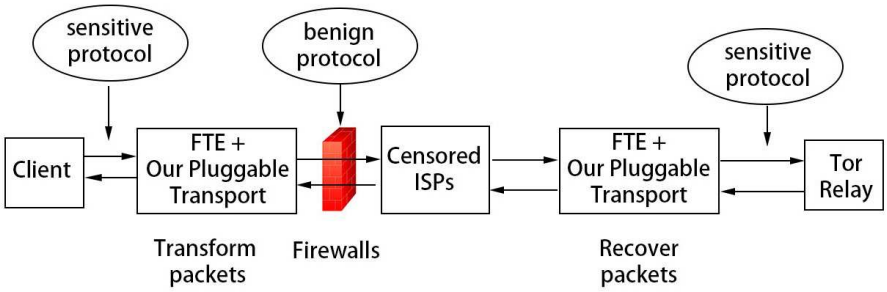


Figure 6.7: System architecture

Table 6.3: Volume of collected campus traffic

Date	Traffic volume
01/03/2015	70.3 G
01/14/2015	66.1 G
01/15/2015	60.8 G
01/16/2015	36.2 G
01/19/2015	32.5 G

develop a Tor PT.

6.4.2 Campus traffic collection and modelling

6.4.2.1 Campus traffic collection

The goal is to make Tor traffic look like universally used benign protocols to evade censorship and surveillance conducted by the local authority. The first step is to locate universally used benign (host) protocols to be mimicked. We collected mirrored network traffic of Clemson University during the Spring semester. Our traffic collection was conducted over 5 workdays in January, 2015. Details of campus traffic we collected are in Table 6.3. We noticed that traffic volume is significantly reduced on 01/19/2015 (Martin Luther King Day), which is reasonable.

For each day listed, we conducted traffic collection from time stamp 23:59:59 of the previous day for 24 hours. We used tcpdump to dump traffic and saved the data in .pcap format. We only collected the first 100 bytes of each packet, since it contained the most important information in the packet header (time stamp, source and destination IP address, protocol and packet length). For each pcap file, we will identify application-layer protocols of the network traffic, and then locate widely used benign protocols that we may want to mimic.

6.4.2.2 Protocol identification tools

Transport-layer protocols, TCP and UDP, cover most network traffic, but labelling a packet as TCP or UDP doesn't provide much information. We are more concerned about what application is being used. So we need to identify application-layer protocols based on the pcap file. To do this, multiple tools are evaluated.

(1) Tshark [176]: Tshark is a command-line version of wireshark, which is a network protocol analyzer tool. It is able to convert decoded pcap file to standard output. Tshark identifies popular protocols using a mixture of port-mappings, heuristics (signatures) and protocol data.

(2) Scapy [176]: Scapy is a powerful interactive packet manipulation program. It also reads pcap files and converts them into human-readable format. Sample output of Scapy is in Figure 2. We parse through the Scapy output and manually map port number to protocols using socket.getservbyport in python. We notice that some protocols we obtained with Scapy and Tshark are different. We will use the more specific protocol. For example, if Tshark identifies it as SSH and Scapy identifies it as TCP, we label the session as SSH.

(3) L7-filter [48]: L7-filter is an open source project which focuses on identifying the application level (layer 7 in the OSI reference model) protocol. It uses regular expressions to identify protocols and works with 125 application-layer protocols. There are several disadvantages of the L7-filter: (a) One has to manually create the fingerprints (regular expressions) for protocols which are not included in the default protocol lists. (b) Some complex patterns require a lot of processing power. (c) L7-filter only works with real-time traffic collection and analysis, and it doesn't work with pcap files or tcpreplay. So we chose not to use L7-filter for protocol identification.

(4) Fl0p [82]: Fl0p was developed by Michal Zalewski in 2006. It is a capable of passive fingerprinting of layer 7 traffic based on the analysis of packet flow characteristics (size ratios, delays, ordering) alone, without the inspection of payload signatures or port numbers. It is a proof-of-concept application and currently supports only TCP, UDP and ICMP. So we choose not to use Fl0p for protocol identification.

(5) Ntopng [113]: Ntopng is a high-speed web-based traffic analysis and flow collection. Ntopng is based on libpcap. Users can use a web browser to navigate through traffic information and

network status by loading from a pcap file.

After investigating 5 working protocol identification tools, we found 3 of them suitable for our use. They are Tshark, Scapy and Ntopng. We will combine the protocol identification results from these applications and produce a potential protocol list that we will consider for mimicking.

6.4.2.3 Campus traffic protocol identification

Table 6.4 is a partial list of the protocols that Tshark/Scapy/Ntopng was able to identify in the campus data flows. We are interested in 5 features of the campus traffic:

- Protocols: unique protocols identified
- Seconds: total seconds that each protocol is used
- Times: occurrence times for each protocol
- Packet size: total packet size for each each protocol
- Volume: Packet size / Seconds

We went through the protocols in the list and found the best candidate protocol to mimic. We considered BACnet-apdu, whcih is a ISO global standard communication protocol for building automation and control networks. But the traffic we collected were limited to Clemson University, which may not represent the normal behavior of BACnet protocol.

A good candidate protocol should be widely used and able to carry two-side payload with reasonable inter-packet delays. We finalized the target protocols as Network Time Protocol (NTP), which is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks.

6.4.2.4 Pattern extraction

There are several versions and modes of NTP. We choose version 4, client-server mode as the target protocol. Figure 6.8 shows the packet fields of a NTP packet. Note that NTP packets have a constant length of 90 bytes.

We concatenated NTP traffic samples with the same patterns one by one to generate an overall timing patterns. Figure 6.9 and 6.10 shows the histograms of the aggregated traffic. And

Table 6.4: Top 100 protocols used in Clemson campus (frequency decreases from left to right and from top to bottom)

capwap-data	http	udp	ncdmirroring
bacnet-apdu	afpovertcp	synapse-nhttp	microsoft-ds
https	tcp	nfs	ipv4
connected	802.11	snmp	capwap-control
asa-appl-proto	esp	webcache	macromedia-fcs
vnc-server	icmp	blp2	modbus/tcp
ldap	commplex-link	ssh	talarian-tcp
ms-wbt-server	rtsp	pppcomp	bootp
sieve-filter	fcp-addr-srvr1	jetdirect	eigrp
kerberos	printer	arp	domain
netbios-ssn	tw-auth-key	crestron-cip	ssdp
xmpp-client	vids-avtp	svrloc	llc
ntp	tproxy	imaps	isakmp
nbns	epmap	swat	tripe
dtlsv1.0	gre	msft-gc	boks
ftp	radan-http	inovaport3	pptp
tftp	hpvirtgrp	srvloc	fmpro-internal
indigo-server	d-cinema-rrp	autodesk-lm	atc-appserver
cvmon	accelenet	res	sd-elmd
nucleus	mps-raft	nrcabq-lm	ismaeasdaplive
slingshot	bmc-perf-agent	net-assistant	hpvroom
stun	nburn id	msl lmd	submission
newbay-snc-mc	smtp	boks servc	cslistener
galaxy7-data	commplex-main	eap	boks servm
ms-sql-s	lldp	tlsv1.2	aequus

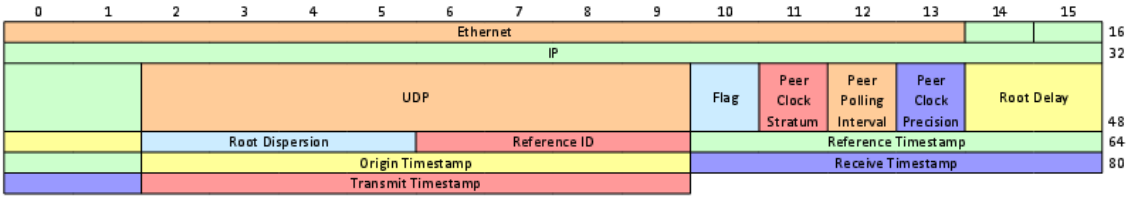
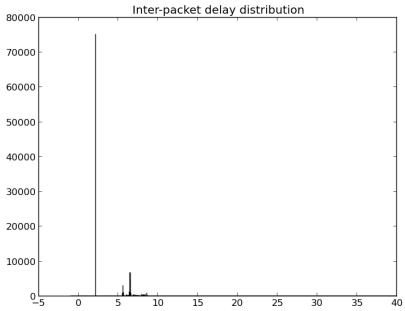
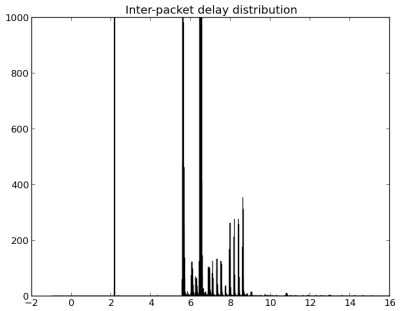


Figure 6.8: NTP packet fields

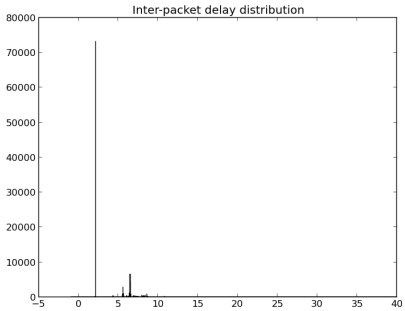


(a) Overall client histogram

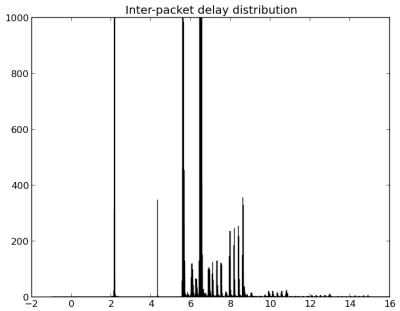


(b) Zoomed overall client histogram

Figure 6.9: Overall timing pattern from client to server of NTP protocol



(a) Overall server histogram



(b) Zoomed overall server histogram

Figure 6.10: Overall timing pattern from server to client of NTP protocol

we used the same method to infer HMMs to represent the timing patterns of NTP. Figure 6.11 and 6.12 show the generated HMMs.

The goal of the project is to transform and obfuscate the source protocol (Tor protocol) so that it will be falsely identified as the target protocol by the censors. Two features are of interest: syntax, and side-channel features (inter-packet time delay and packet size). We transformed the Tor protocol into NTP and Minecraft. For both, revised FTE will first be used to transform Tor into the target syntax. Then we cut the FTE output into different chunks based on the packet size of the target protocol, and change the timing information based on the inter-packet time delay of the target protocol.

For NTP, it is easier because the packet size is constant (90 bytes). By cutting the FTE output into chunks of 90 bytes, we have packets following the NTP syntax, which would be identified as NTP by network monitoring tools or protocol identification tools. Then we send the packets based

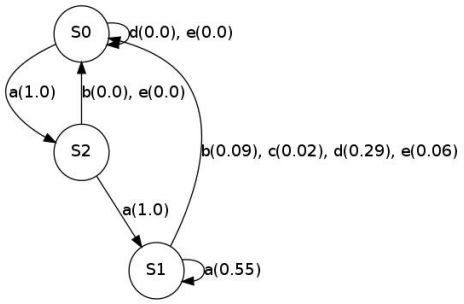


Figure 6.11: HMM model inferred from NTP (from client to server)

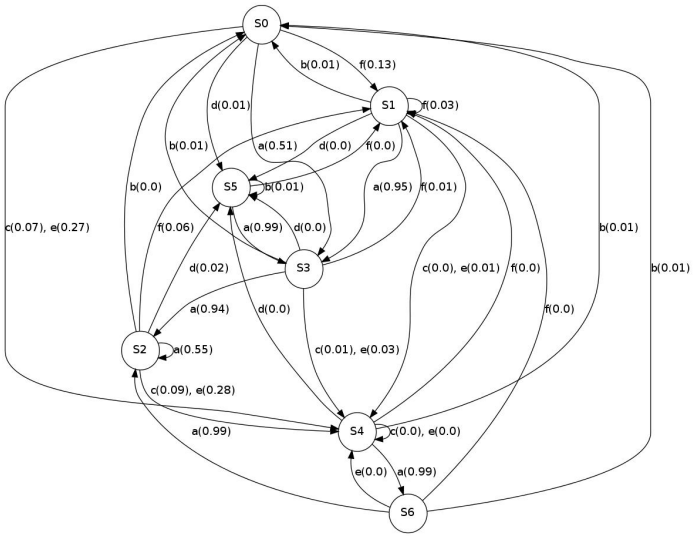


Figure 6.12: HMM model inferred from NTP(from server to client)

Table 6.5: Minecraft traffic details

Label	File size (MB)	Port Number	Total number of packets	Number of packets from server to client	Number of packets from client to server
1	121.2	46444	612569	396372	216197
2	53.3	46990	373441	234886	138555
3	33.1	47179	111901	63322	48579
4	105.5	43532	487862	273125	214737
5	61.2	33010	301873	176394	125479
6	14.9	33213	162979	32892	130087

on the inter-packet delays extracted from the HMM of NTP, which will guarantee the statistical similarity between the transformed traffic and the real NTP traffic.

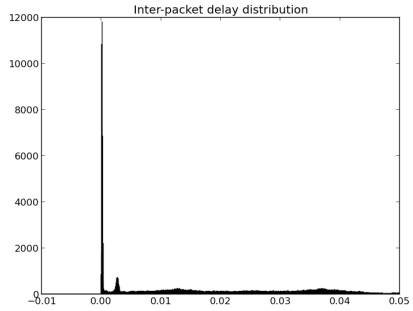
6.4.3 Game traffic collection and modelling

In order to choose a good game protocol to mimic, we were thinking of Defense of the Ancients (DotA) and League of Legends (LoL). Both are multiplayer online battle arena video games and have so many players. But by analyzing their traffic, there are more than 100 IP addresses used by a single player, which makes it difficult to build a client-server model. Minecraft is a popular server-dependent video game about placing blocks to build anything you can imagine. People play Minecraft on different servers, where players send building information to the server, and retrieve map data from the server. So it is a good model to carry client-server communication. By observing Minecraft traffic, we verified the possibility of using Minecraft as the cover protocol.

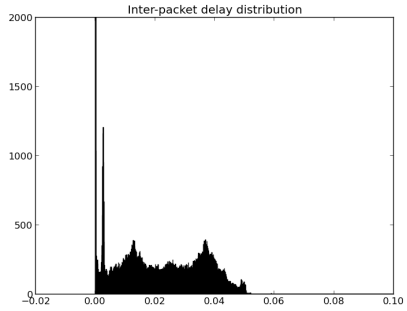
6.4.3.1 Minecraft traffic collection

Different Minecraft players play on different servers, and the achievements of one player are only kept on a specific server. Since all communication is between a player and a server, it is a good candidate to carry client-server communication for Tor. We setup our own Minecraft server (130.127.24.141:9090) in Clemson University and ask several students to play on it. Since different game sessions use different ports, we use port number to differentiate game sessions. A total of 6 game sessions are recorded (Table 6.5).

By observing the Minecraft traffic, we notice that all packets use TCP protocols with a random higher port. It fits the three-way handshake routines in TCP protocol. Note that payload (data) can be sent in both ways, which makes it easier to carry two-way traffic in Tor communication.



(a) One example histogram



(b) Zoomed histogram

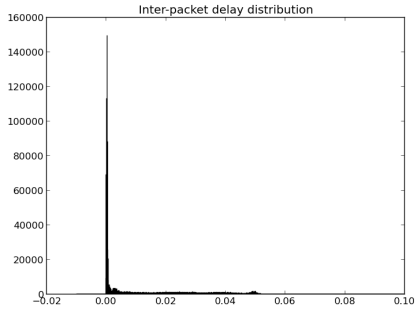
Figure 6.13: Timing pattern of traffic 2 from client to server

6.4.3.2 Pattern extraction

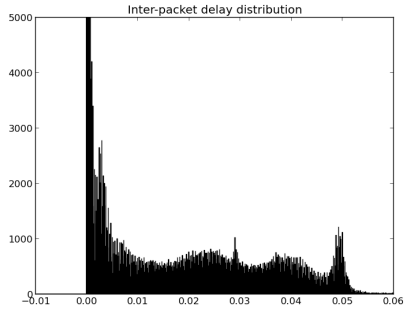
We collected 6 pieces of Minecraft traffic and drew the histogram to find the similarities between them. All histograms, regardless of traffic direction(from client to server, or from server to client), show the similar tendency in Figure 6.13. Figure 6.13(a) shows the histogram of one traffic sample from client to server, and Figure 6.13(b) shows the zoomed histogram with frequency less than 2000. From the figure, we can see that the most frequent timing (inter-packet delay) is a little larger than 0.00 second, and almost all timings are less than 0.05 second, which are true for all traffic samples. The difference between different traffic samples lies in the bumps after 0.01 second. The result shows that Minecraft has a large traffic volume, but a short inter-packet delay, which makes it a perfect carrier of Tor traffic.

Since we have 6 pieces of Minecraft traffic and they have similar tendency in traffic patterns, we concatenate them on by one to generate an overall timing patterns in Figure 6.14 and 6.15. Figure 6.14 is the aggregated traffic from client to server, and Figure 6.15 is the aggregated traffic from server to client.

Based on the histogram, we inferred an HMM using the method [134]. HMM is a mathematical representation of the timing pattern of the traffic. With the HMM, we can generate packets based on the pre-defined inter-packet delays. To do that, we discretize the timings into symbols. Then an HMM is built by calculating the transition probability between any two symbols. To generate a string of symbols with an HMM, one picks a random start state and goes through the states (throw a dice on each transition to decide the path). If we map the symbols back to timings, we get the inter-packet delays to send out the packets. This method can mimic the side-channel information

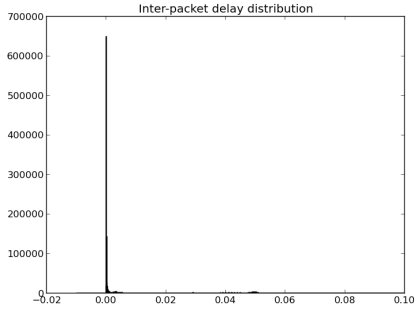


(a) Overall client histogram

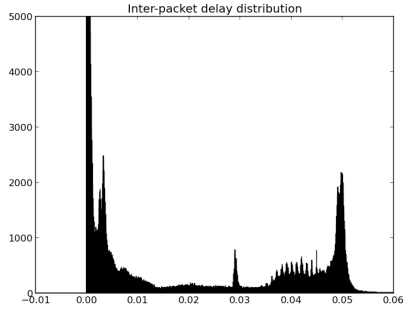


(b) Zoomed overall client histogram

Figure 6.14: Overall timing pattern from client to server of Minecraft



(a) Overall server histogram



(b) Zoomed overall server histogram

Figure 6.15: Overall timing pattern from server to client of Minecraft

of any target protocols. The inferred HMM models based on Minecraft traffic is in Appendix (Figure 6.16 and 6.17).

Different from transforming Tor to NTP, it is fairly hard to transform Tor to Minecraft. The reason is that Minecraft are basically TCP and UDP packets containing binary gaming data, and the packet size varies from packet to packet. This makes it impossible to cut the FTE output into chunks of constant size. We come up with a method to model the packet size and timing together, that is to infer a two-dimension HMM where packet size is one dimension and timing is the other. To achieve that, we parse through the packet sizes and timing information separately of all Minecraft packets from the players to the game server, and draw a histogram in Figure 6.18(a) (packet size) and Figure 6.18(b) (timing info). We can see that 6 bins can cover the packet size, and 2 bins can cover the timing info. This means that 12 bins can cover Minecraft traffic if we model them together. Table 6.6 shows the bins and the representations of the two-dimension HMM.

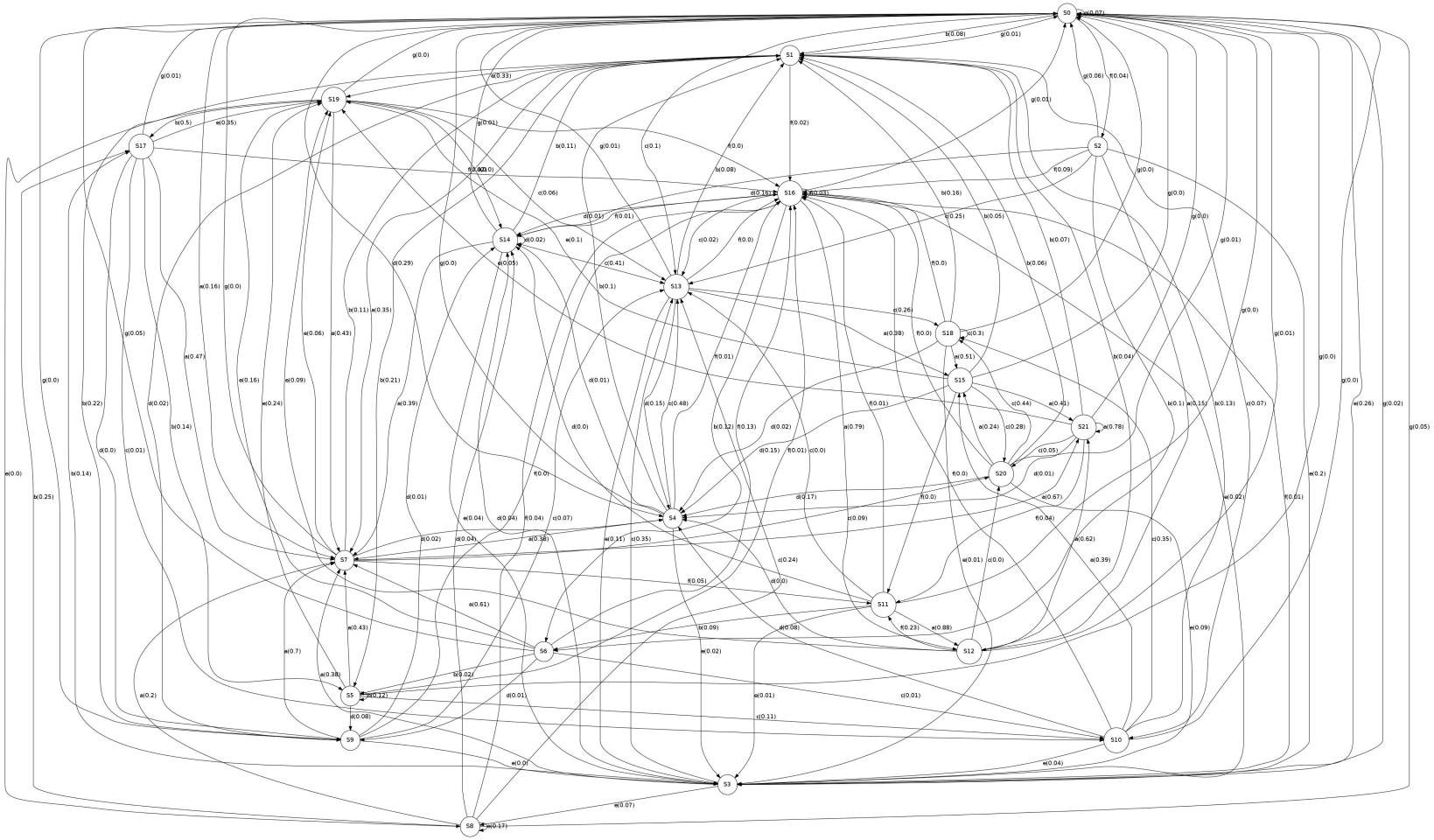


Figure 6.16: HMM model inferred from Minecraft (from client to server)

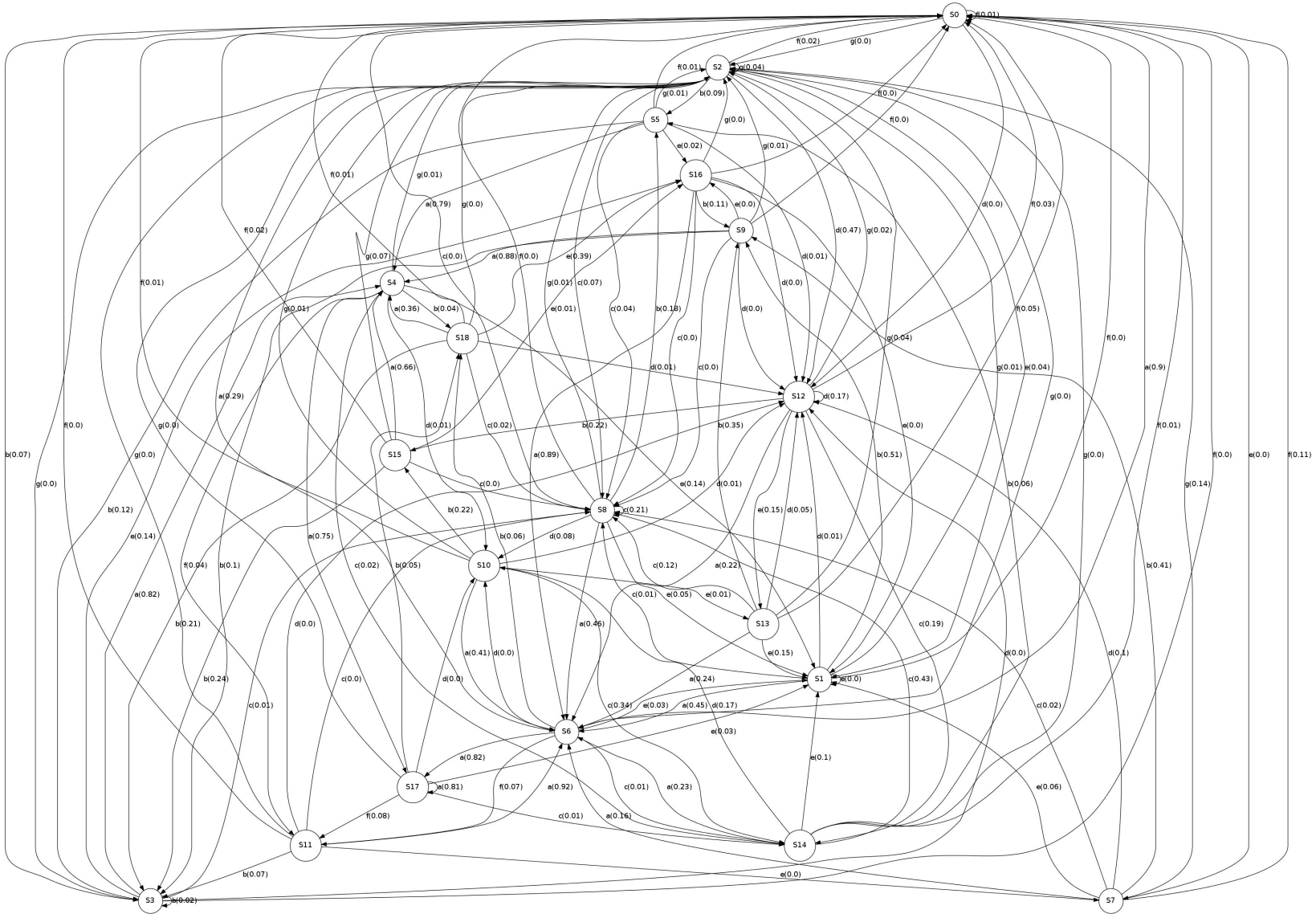
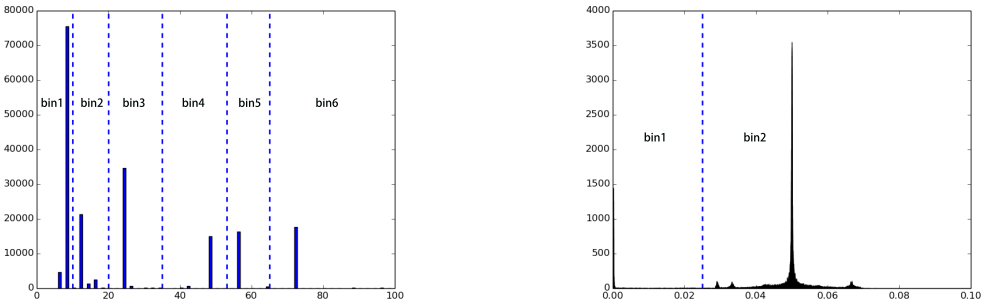


Figure 6.17: HMM model inferred from Minecraft (from server to client)



(a) Packet size with 6 bins

(b) Timing histogram with 2 bins

Figure 6.18: Pakcet size and timing histogram

Table 6.6: Bins and representations of the two-dimension HMM

Bin No.	Bin symbol	Inter-packet time delay	packet size range in bytes
1	a	< 0.025	< 10
2	b	< 0.025	10 - 20
3	c	< 0.025	20 - 35
4	d	< 0.025	35 - 53
5	e	< 0.025	53 - 65
6	f	< 0.025	> 65
7	g	> 0.025	< 10
8	h	> 0.025	10 - 20
9	i	> 0.025	20 - 35
10	j	> 0.025	35 - 53
11	k	> 0.025	53 - 65
12	l	> 0.025	> 65

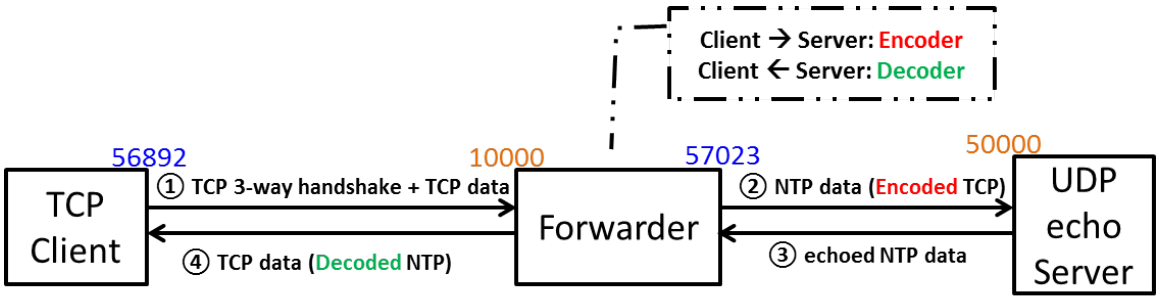


Figure 6.19: A forwarder that encodes TCP and decodes NTP packets

6.4.4 Results and analysis

As a proof of concept to illustrate the idea, we designed a forwarder that forwards traffic between a TCP client and UDP (NTP) echo server. All traffic that goes through the forwarder from client to server will be encoded from TCP to NTP, and all traffic that goes through the forwarder from server to client will be decoded from NTP to TCP. Figure 6.19 shows the procedures of a TCP-NTP communication.

- Step 1: TCP client sends random data to the forwarder (source port:56892, destination port: 10000);
- Step 2: The forwarder encodes the TCP data into NTP packets and sends them to UDP echo server (source port: 57023, destination port: 50000);
- Step 3: The UDP echo server sends the received message back to the forwarder (source port: 50000, destination port: 57023);
- Step 4: The forwarder decodes NTP packets into TCP data and sends them to the TCP client (source port:10000, destination port: 56892).

We set up tcp client, forwarder, and UDP echo server on the same machine and use the loopback (IP address: 127.0.0.1) for the communication. Figure 6.20 shows the screenshot of encoded NTP traffic. We can see that TCP packets are successfully transformed into NTP packets. And the packets can be decoded into correct NTP fields with Wireshark.

One issue to notice with the transformation is that the timestamp in NTP packet fields looks weird. Figure 6.21 shows the packet fields in a real NTP packet. In a real NTP packet, there is a very small difference (less than 2 minutes) among reference timestamp, origin timestamp, receive timestamp, and transmit timestamp. However in a fake NTP packet, there is a much larger difference among them, and the difference is up to 5 days. How NTP works is to calculate the offset from the reference timestamp using those timestamps and adjust the clock accordingly [11]. As long as the difference is no more than 68 years, NTP works fine. If NTP is running in the background, the difference should be quite small. However, the initial NTP packets can have a larger difference. And a mis-functioned NTP server can have similar patterns to what we generated.

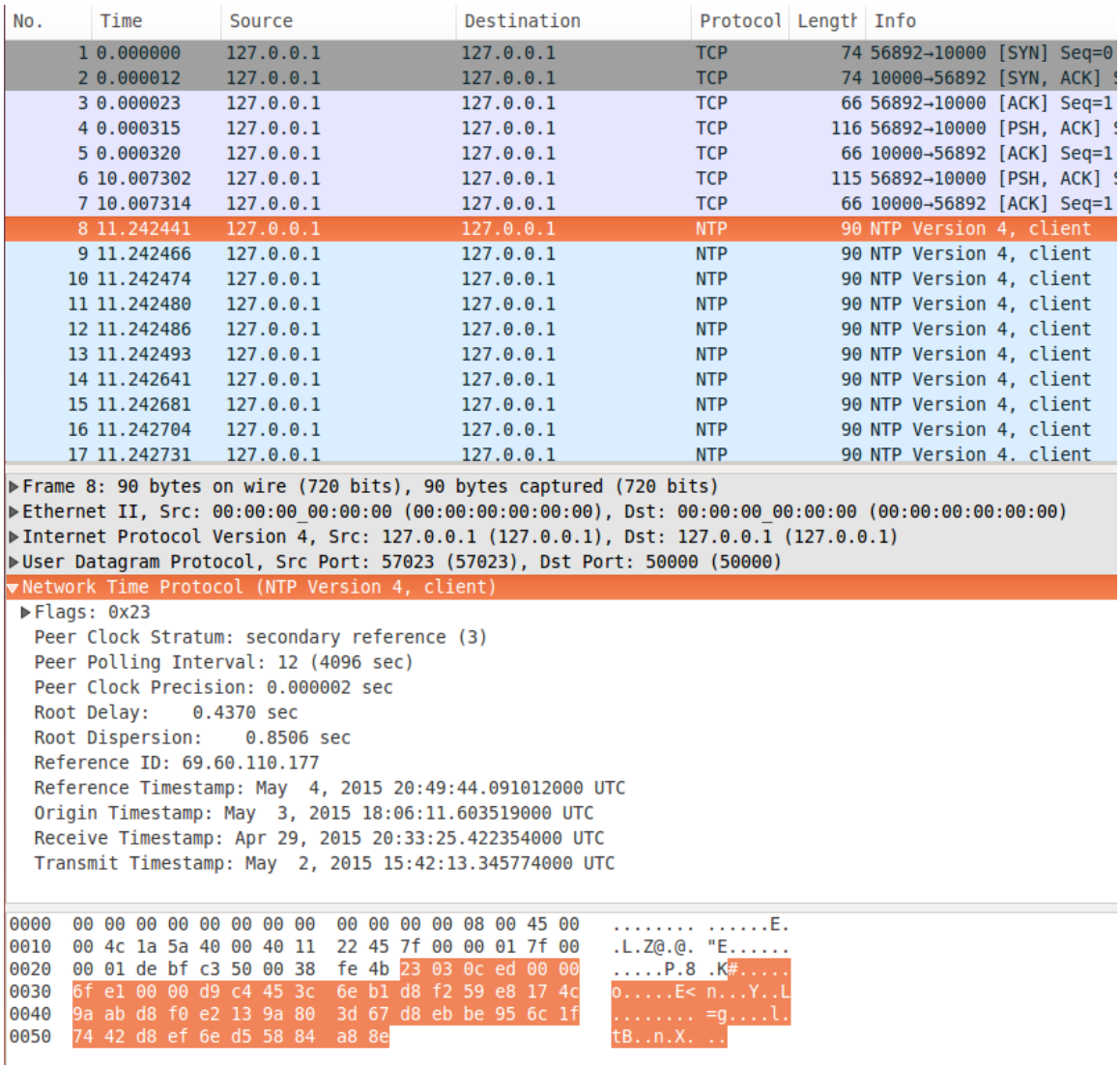


Figure 6.20: Screenshot of encoded NTP traffic (step 2)

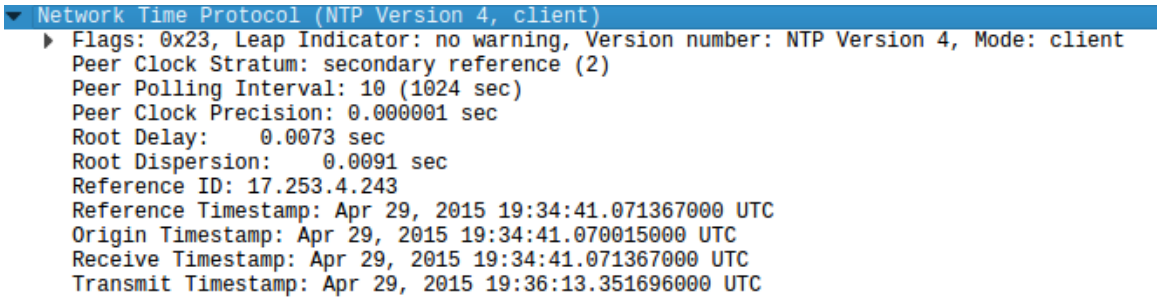


Figure 6.21: Packet fields of a real NTP packet

6.5 Summary

In this chapter, we provided two applications that illustrate the proposed protocol obfuscation technique. It can transform an arbitrary sensitive protocol to any target protocol. The choice of the target protocol can be tricky, but a good choice of target protocol can guarantee good throughput of the communication. The protocol obfuscation technique is a combination of revised FTE (handle syntax) and SCM (handle side-channel features). Experiment results show that it works well in protocol transformation and mimicry.

Chapter 7

Conclusions

Botnets cause enormous damage to our property and pose threats to our daily life. In this work, we proposed several ideas on how to apply botnet techniques (fast-flux and DGA) to the legitimate use. With our solutions, we can benefit from the botnet techniques for secure and private communicaton, as well as being pro-active in defending botnets.

We developed two novel DGAs using hidden Markov models (HMMs) and Probabilistic Context-Free Grammars (PCFGs). Experiments show that our DGAs are effective in thwarting the current detection techniques (KL, ED and JI) and detection systems (BotDigger and Pleiades). We model the competition between security personnel (detection techniques) and botmasters (DGAs) as a Two-Person Zero-Sum (TPZS) game. This provides a methodology for selecting optimal strategies for both sides, given different detection techniques and DGAs. Also, we developed two HMM-based DGA detection, which outperformed existing detection KL/ED/JI in detecting both existing and previously unknown DGAs.

Then we provided two applications using botnet technologies. The first application implemented a distributed proxy system using botnet fast-flux to evade censorship and surveillance in West Africa. The second application designed a covert data transport protocol using botnet DGA to counteract network analysis. To our best knowledge, this is the first time to apply botnet technologies to real-life applications for normal Internet users.

Lastly, we proposed a protocol obfuscation technique that can transform an arbitrary sensitive protocols to any target protocols. As examples, we provided two applications for offline and online transformation. The result is very promising.

This work focuses on botnet research, which includes understanding botnet, developing botnet detection and evasion techniques, and applying botnet technologies to real-life applications. We hope this will help security personnel have a better understanding of botnet phenomenon and finally take down the botnets.

At the same time, botnets are evolving fast and security personnal cannot stop botnets once for all. Although intrusion detection systems (IDSs) or most detection techniques can help in detecting network anomaly based on existing patterns and signatures, they cannot detect zero-day botnets. Moreover, it is expensive and incredibly hard to develop a bulletproof IDS that can detect all existing botnets. From our work, we conclude that network traffic is a malleable and artificial construct. Although existing patterns are easy to detect and characterize, they are also subject to modification and mimicry. This means that we can construct transducers to make any communication pattern look like any other communication pattern. It is a fact that we need to accept and use as best we can. We hope this work can alert security personnel in designing IDS or other detection technique to be proactive in fighting against botnets.

Appendices

Appendix A How to obtain IPV4 domain names

To obtain domain names for the IPv4 space, we used the Sonar project [159], which provides a database of IPv4 reverse DNS records (updated at 2014-03-28). We downloaded the domain records of the entire IPv4 space as legitimate domain names. Since Botmasters register DGA-generated domain names to communicate with bots, DGAs should not generate domains that conflict with existing records. So we deleted the following patterns in IPv4 domain names to avoid registered domain names:

(1) IP addresses (XXX-XXX-XXX-XXX where XXX is a number in the range 0-255)

(2) Repeated patterns with a large number of records (ip-X, iptv-X, dsl-X, ti-X, host-X, hostX ull-X, ntl-X, link-X, OL-X, ss-X, dhcp-X, dhcpX, sl-X, i-d-X, user-X, dyn-X, static-X,adsl-X, ppp-X, softbank-X, Dinamico-X, -ipbfp-X, AWBLnew-X, ipX, 0xX, client-X), where X is a number or a IP address

Also, botmasters use fast-flux to change the mapping between domain names and IP addresses to evade detection. They often register free domain names from dynamic DNS websites. Most of those websites don't accept domain with special characters other than letters, numbers and hyphen, so we deleted domain names with those patterns.

Appendix B How to obtain a *number+hyphen* list to generate PCFG training sets

PCFG requires training sets containing information from IPv4 domain names. A hyphenation algorithm [88] was used to extract syllables from IPv4 domain names, but it lacked numbers and hyphens. In order to have a reasonable *number + hyphen* list, we went through the following steps:

- Step1: Parse all IPv4 domains by separating alphabetic and numerical characters, and delete repeated entries in each list. The aphabet list is called List 1, and the numerical list is called List 2. For example, we separated $789abc-123def$ as 789, abc, 123, and def.
- Step 2: Regenerate a *number + hyphen* list (called List 3) by adding an optional hyphen to each entry from List 2 in Step 1.

To generate domain names for PCFG training sets, we calculated the probability of there being a number in an IPv4 domain name is 0.567. Then we generated domains by choosing entries from List 1 and List 3 in a way that guarantees the probability of a number being in the generated domain name is also 0.567.

Appendix C Curriculum Vitae

• Contact Information

Email: fu2@g.clemson.edu

Mobile Phone: 864-207-0457

Address: 150 Briar Ln, Central, SC 29630

• Education Background

Clemson University GPA: 4.0/4.0

Ph.D candidate in Network Security (Minor: Statistics) Expected Aug 2017

Department of Electrical and Computer Engineering Aug. 2012- Present

East China University of Science and Technology, Shanghai, China GPA: 3.7/4.0

Bachelor of Engineering in Electrical Engineering and Automation Sep. 2008- May. 2012

School of Information Science and Engineering

• Honors & Awards

The third International Conference on Malicious and Unwanted Software (Malware 2016)

Capture the Flag (CTF) Competition, Second Place 2016

Best Paper Award in Cyber and Information Security Research Workshop (CISR15) 2015

Outstanding Graduate Student in College of Engineering and Science 2013-2014

Palmetto Cyber Defense Competition, First Place (team leader) 2013,2014

Third prize of East Division in ADI University Design Competition(UDC)(7/460) 2010

National Scholarship, Education Department of China (Top 1%) 2010

Top Prize in Comprehensive Curriculum Scholarship, ECUST (1/81) 2009

Second Prize in Mathematics Competition, ECUST 2009

• Academic Experience

Security of Cyber-Physical Systems with Connected Vehicles Sep. 2016 - present

- Developed the highway automotive platooning system with centralized/decentralized control

- Designed Kalman filter to compensate the white sensor/plant noise
- Working on analyzing the security of platooning system with the game theory models
- Designing controllers that are resilient to the cyber-attacks

Research of botnet Domain Generation Algorithms (DGAs) Jan. 2013 - May 2017

- Proposed 2 novel DGAs using Hidden Markov Models (HMMs) and Probabilistic Context-Free Grammars (PCFGs)
- Evaded existing detection techniques very well
- Evaluated DGA performance with Game Theory models

Internet Freedom Project for West Africa Sep. 2012 - May 2015

- Developed a distributed proxy system to evade censorship and surveillance
- Coded the proxy system in Linux, Windows and Android
- Helped train journalists and NGOs with circumvention technology in Paris and Abidjan

Development of Tor Pluggable Transport Layer Nov. 2014 - Dec. 2015

- Analyzed the mirrored campus traffic for application-layer protocols
- Building Hiddem Markov models of network protocols
- Designing a Tor pluggable Transport Layer to evade censorship and surveillance

Research of Eco-system of botnets Sep. 2012 - Oct. 2014

- Looked into the weakness of Two-Factor-Authentication (2FA) banking system
- Designed a structure to break 2FA in iPhone-based botnets
- Studied the interactions in botnet markets

• Publications

■ **Yu Fu**, Lu Yu, Oluwakemi Hambolu, Ilker Ozcelik, Benafsh Husain, Jingxuan Sun, Karan Sapra, Dan Du, Christopher Tate Beasley, and Richard R. Brooks. *Stealthy Domain Generation Algorithms.* IEEE Transactions on Information Forensics and Security 12, no. 6 (2017): 1430-1443.

■ **Yu Fu**, Zhe Jia, Lu Yu, Xingsi Zhong, and Richard R. Brooks. A covert data transport protocol

2016 11th International Conference on Malicious and Unwanted Software (MALWARE). 2016 11th International Conference on, pp. 1-8. IEEE, 2016.

■ **Yu Fu**, Benafsh Husain and Richard R. Brooks. Analysis of Botnet Counter-Counter-Measures. *Proceedings of the 10th Annual Cyber and Information Security Research Conference.* 2014.

■ Xingsi Zhong, **Yu Fu**, Lu Yu, Richard R. Brooks, and G Kumar Venayagamoorthy. Stealthy malware traffic-Not as innocent as it looks. *2015 10th International Conference on Malicious and Unwanted Software (MALWARE).* 2015.

■ Ilker Ozcelik, **Yu Fu**, and Richard R. Brooks. DoS Detection is Easier Now. *The Second GENI Research and Educational Experiment Workshop (GREE2013).* 2013.

■ Xingsi Zhong, Paranietharan Arunagirinathan, Richard R. Brooks, Ganesh Kumar Venayagamoorthy, Lu Yu and **Yu Fu**. Side Channel Analysis of Multiple PMU Data in Electric Power Systems. *2015 Power Systems Conference.* 2015.

■ Richard R. Brooks, Oluwakemi Hambolu, Paul Marusich, **Yu Fu**, Saiprasad Balachandran. Creating a Tailored Trustworthy Space for Democracy Advocates using Hostile Host. *CSIIRW 13: Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop.* Jan. 2013

■ Juan Deng, Lu Yu, **Yu Fu**, O. Hambolu, Richard R. Brooks. Security and Data Provenance of Modern Automobiles, Data Analytics for Intelligent Transportation Systems. Elsevier S&T Books. To be published

■ **Yu Fu**, Lu Yu, and Richard Brooks. Domain Generation Algorithms (DGAs) Detection using Hidden Markov Models (HMMs). *IEEE Transactions on Information Forensics & Security.* Under review.

■ Richard R. Brooks, Lu Yu, **Yu Fu**, Oluwakemi Hambolu, Narasimhamurthy Rampuraswathanarayana, John Gaynard, Julie Owono, Archippe, and Felix Blanc. Internet Freedom in West Africa. *Communications of the ACM.* Under review.

■ Oluwakemi Hambolu, Lu Yu, **Yu Fu**, and Richard R. Brooks. Privacy Preserving Cardinality Statistics. *IEEE Transactions on Systems, Man and Cybernetics: Systems.* Under review.

■ **Yu Fu**, Lu Yu, and Richard R. Brooks. A survey of Botnet Taxonomy and Detection Techniques. In preparation.

■ **Yu Fu**, Lu Yu, Jon Oakley, and Richard R. Brooks. Analysis of Botnet Domain Generation Algorithms (DGAs) family. In preparation.

■ **Yu Fu**, Kuilin Chen, Ziqiang Huang, and Xiaoqiang Wang. Design of Environment Parameter Monitor System Based on Zigbee Wireless Network. *Industrial Intelligentialization.*

• Industrial Experience

Emerson Process Management Co. Ltd. Shanghai, China

Engineering Intern in Rosemount (RMT) Department July 2011 - Sep. 2011

- Helped choose types of measuring instruments for customs and engineers
- Revised and translated datasheets of Rosemount instruments
- Summarized results of surveys from customs and wrote relevant reports

• Extra-Curricular Activities

Treasurer of Clemson University Badminton Club (CUBC) 2013-2014

Member of Clemson University Cyber Defense Team (CUCD) 2013

Group Leader of Shanghai EXPO volunteers May 2010

• Skills & Interests

Languages: Fluent English, Native Mandarin

Programming: Python, Java, Matlab, C

Interests: Keyboard player in band *GALLERY*

Bibliography

[1] Iran reportedly blocking encrypted internet traffic. http://arstechnica.com/tech-policy/2012/02/iran-reportedly-blocking-encrypted-internet-traffic/. [Last visited: 05-July-2016].

[2] Raihana Syahirah Abdullah, Mohd Faizal Abdollah, Zul Azri Muhamad Noh, Mohd Zaki Masud, Siti Rahayu Selamat, Robiah Yusof, and Universiti Teknikal Malaysia Melaka. Revealing the criterion on botnet detection technique. *IJCSI International Journal of Computer Science Issues*, 10(2):208–215, 2013.

[3] Esraa Alomari, Selvakumar Manickam, BB Gupta, Shankar Karuppayah, and Rafeef Alfaris. Botnet-based distributed denial of service (ddos) attacks on web servers: Classification and art. *arXiv preprint arXiv:1208.0403*, 2012.

[4] Chema Alonso and Manu "The Sur". Owning bad guys {& mafia} with javascript botnets, 2012. http://media.blackhat.com/bh-us-12/Briefings/Alonso/BH_US_12_Alonso_Owning_Bad_Guys_WP.pdf.

[5] Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann, and Herbert Bos. Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus. In *MALWARE*, pages 116–123. IEEE, 2013.

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

[7] Axelle Apvrille. Cryptography for mobile malware obfuscation. In *RSA Conference Europe*, 2011.

[8] Sajjad Arshad, Maghsoud Abbaspour, Mehdi Kharrazi, and Hooman Sanatkar. An anomaly-based botnet detection approach for identifying stealthy botnets. In *Computer Applications and Industrial Electronics (ICCAIE), 2011 IEEE International Conference on*, pages 564–569. IEEE, 2011.

[9] Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. A survey of botnet technology and defenses. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pages 299–304. IEEE, 2009.

[10] Paul Barford and Vinod Yegneswaran. An inside look at botnets. In *Malware Detection*, pages 171–191. Springer, 2007.

[11] L. Frank Baum. Ntp timestamp calculations, 2012. https://www.eecis.udel.edu/~mills/time.html.

[12] Alpha Bay. Alphabay market, 2017 (Last visited). http://pwoah7foa6au2pul.onion/index.php.

[13] Alan Beale. 12 dicts introduction - the 3esl list, 2003. http://wordlist.aspell.net/12dicts-readme/.

[14] Brigitte Bigi. Using kullback-leibler distance for text categorization. In *Proceedings of the 25th European Conference on IR Research*, ECIR'03, pages 305–319, Berlin, Heidelberg, 2003. Springer-Verlag.

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

[16] Leyla Bilge, Sevil Sen, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. Exposure: a passive dns analysis service to detect and report malicious domains. *ACM Transactions on Information and System Security (TISSEC)*, 16(4):14, 2014.

[17] Hamad Binsalleeh, Thomas Ormerod, Amine Boukhtouta, Prosenjit Sinha, Amr Youssef, Mourad Debbabi, and Lingyu Wang. On the analysis of the zeus botnet crimeware toolkit. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, pages 31–38. IEEE, 2010.

[18] Brandan Blevins. Financial malware focuses on hiding malicious traffic, localization. http://searchsecurity.techtarget.com/news/2240212531/Financial-malware-focuses-on-hiding-malicious-traffic-localization, 2014. [Last visited: 06-Aug-2015].

[19] Taylor L Booth and Richard A Thompson. Applying probability measures to abstract languages. *Computers, IEEE Transactions on*, 100(5):442–450, 1973.

[20] Kenton Born and David Gustafson. Detecting dns tunnels using character frequency analysis. *arXiv preprint arXiv:1004.4358*, 2010.

[21] Brave New Software. Lantern open internet for everyone, 2016 (last visited). https://getlantern.org/.

[22] Richard R Brooks, Oluwakemi Hambolu, Paul Marusich, Yu Fu, and Saiprasad Balachandran. Creating a tailored trustworthy space for democracy advocates using hostile host. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, page 50. ACM, 2013.

[23] Richard R Brooks, Jason M Schwier, and Christopher Griffin. Behavior detection using confidence intervals of hidden markov models. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(6):1484–1492, 2009.

[24] Lu Chen, Jason M. Schwier, Ryan M. Craven, Lu Yu, Richard. R. Brooks, and Christopher Griffin. A normalized statistical metric space for hidden markov models. 43(3):806 – 819, 2013.

[25] Zhiyi Chi. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160, 1999.

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

[27] Noam Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, 1956.

[28] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of the USENIX SRUTI Workshop*, volume 39, page 44, 2005.

[29] Heather Crawford and John Aycock. Kwyjibo: automatic domain name generation. *Software: Practice and Experience*, 38(14):1561–1567, 2008.

[30] Heather Crawford and John Aycock. Kwyjibo: Automatic domain name generation. *Softw. Pract. Exper.*, 38(14):1561–1567, November 2008.

[31] David Dagon, Guofei Gu, Christopher P Lee, and Wenke Lee. A taxonomy of botnet structures. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 325–339. IEEE, 2007.

[32] Neil Daswani and Michael Stoppelman. The anatomy of clickbot. a. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 11–11. USENIX Association, 2007.

[33] Christian J Dietrich, Christian Rossow, Felix C Freiling, Herbert Bos, Maarten Van Steen, and Norbert Pohlmann. On botnets that use dns for command and control. In *Computer Network Defense (EC2ND), 2011 Seventh European Conference on*, pages 9–16. IEEE, 2011.

[34] Kevin P. Dyer. fteproxy. `https://fteproxy.org`, 2013. [Last visited: 06-Aug-2015].

[35] Kevin P. Dyer. LibFTE 0.1.0. `https://github.com/kpdyer/libfte`, 2015. [Last visited: 06-Aug-2015].

[36] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 61–72, New York, NY, USA, 2013. ACM.

[37] Sean R Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3):361–365, 1996.

[38] YÜCE Emre. A literature survey about recent botnet trends. 2011.

[39] Verizon Enterprise. 2011 data breach investigations report, 2011. `http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigations-report-2011_en_xg.pdf`.

[40] Meisam Eslahi, Habibah Hashim, and NM Tahir. An efficient false alarm reduction approach in http-based botnet detection. In *Computers & Informatics (ISCI), 2013 IEEE Symposium on*, pages 201–205. IEEE, 2013.

[41] Greg Farnham and A Atlasis. Detecting dns tunneling. *SANS Institute InfoSec Reading Room*, pages 1–32, 2013.

[42] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A survey of botnet and botnet detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*, pages 268–273. IEEE, 2009.

[43] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Fairuz Amalina, Rauf Ridzuan Maarof, and Shahaboddin Shamshirband. A study of machine learning classifiers for anomaly-based mobile botnet detection. *Malaysian Journal of Computer Science*, 26(4), 2014.

[44] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Dan Boneh, Roger Dingledine, and Phil Porras. Evading censorship with browser-based proxies. In *Privacy Enhancing Technologies*, pages 239–258. Springer, 2012.

[45] Dennis Fisher. New version of kelihos botnet appears, 2013. `http://threatpost.com/new-version-kelihos-botnet-appears-021113/77509`.

[46] G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[47] Brian Foster. Three reasons why botnet takedowns are ineffective, 2012. `https://blog.damballa.com/archives/2195`.

[48] Clear Foundation. Application layer packet classifier for linux: L7-filter. `http://l7-filter.sourceforge.net/`, 2009. [Last visited: 06-Aug-2015].

[49] Yu Fu, Benafsh Husain, and Richard R Brooks. Analysis of botnet counter-counter-measures. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, page 9. ACM, 2015.

[50] Yu Fu, Zhe Jiay, Lu Yu, Xingsi Zhong, and Richard Brooks. A covert data transport protocol. In *Malicious and Unwanted Software (MALWARE), 2016 11th International Conference on*, pages 1–8. IEEE, 2016.

[51] Yu Fu, Lu Yu, Oluwakemi Hambolu, Ilker Ozcelik, Benafsh Husain, Jingxuan Sun, Karan Sapra, Dan Du, Christopher Tate Beasley, and Richard R Brooks. Stealthy domain generation algorithms. *IEEE Transactions on Information Forensics and Security*, 12(6):1430–1443, 2017.

[52] Yu Fu, Lu Yu, Oluwakemi Hambolu, Ilker Ozcelik, Benafsh Husain, Jingxuan Sun, Karan Sapra, Dan Du, Christopher Tate Beasley, and Richard R Brooks. Stealthy domain generation algorithms. *IEEE Transactions on Information Forensics and Security*, 12(6):1430–1443, 2017.

[53] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014.

[54] Sebastián García, Alejandro Zunino, and Marcelo Campo. Survey on network-based botnet detection methods. *Security and Communication Networks*, 7(5):878–903, 2014.

[55] Tim Greene. Oops. microsoft takes down some researchers' servers along with citadel botnet sites, 2013. http://www.networkworld.com/news/2013/061013-microsoft-researcher-citadel-botnet-270657.html?page=1.

[56] Network Working Group. Internet relay chat protocol, 1993. https://tools.ietf.org/html/rfc1459#section-1.

[57] Network Working Group. Hypertext transfer protocol – http/1.1, 1999. http://tools.ietf.org/html/rfc2616.

[58] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX Security Symposium*, volume 5, pages 139–154, 2008.

[59] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Usenix Security*, volume 7, pages 1–16, 2007.

[60] Shoufu Hagen, Josiahand Luo. Why domain generating algorithms (dgas)? `http://blog.trendmicro.com/domain-generating-algorithms-dgas/`, Aug. 18, 2016.

[61] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[62] Theodore E Harris. *The theory of branching processes*. Courier Dover Publications, 2002.

[63] Wilbert Jan Heeringa. *Measuring dialect pronunciation differences using Levenshtein distance*. PhD thesis, University Library Groningen][Host], 2004.

[64] Ayelet Heyman. First spyeye attack on android mobile platform now in the wild, 2011. `http://www.trusteer.com/blog/first-spyeye-attack-android-mobile-platform-now-wild`.

[65] Honeynet. Phishing technique three - phishing using botnets, 2005. http://www.honeynet.org/book/export/html/92.

[66] Yung-Tsung Hou, Yimeng Chang, Tsuhan Chen, Chi-Sung Laih, and Chia-Mei Chen. Malicious web content detection by machine learning. *Expert Systems with Applications*, 37(1):55–60, 2010.

[67] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 65–79. IEEE, 2013.

[68] Freedom House. Freedom of the press 2016, 2016. `https://freedomhouse.org/report/freedom-press/freedom-press-2016`.

[69] Ohio State University HPCS Lab. System/application designs, optimizations and implementations on overlay networks, 2006. http://www.cse.ohio-state.edu/hpcs/WWW/HTML/internet-p2p.html.

[70] Ching-Hsiang Hsu, Chun-Ying Huang, and Kuan-Ta Chen. Fast-flux bot detection in real time. In *International Workshop on Recent Advances in Intrusion Detection*, pages 464–483. Springer, 2010.

[71] Fu-Hau Hsu, Chang-Kuo Tso, Yi-Chun Yeh, Wei-Jen Wang, and Li-Han Chen. Browserguard: A behavior-based solution to drive-by-download attacks. *IEEE Journal on Selected areas in communications*, 29(7):1461–1468, 2011.

[72] IEEE. C37.118.2-2011 - IEEE standard for synchrophasor data transfer for power systems. pages 1–53, Dec 2011.

[73] Psiphon Inc. Psiphon - beyond borders, 2016. `https://psiphon3.com/en/index.html`.

[74] InformationWeek. Fast flux botnet nets fraudsters $78 million, 2012. http://www.informationweek.com/attacks/fast-flux-botnet-nets-fraudsters-$78-million/d/d-id/1107073?

[75] Intrnet Without Borders. Your private access to the open internet., 2017 (last visited). `https://www.uproxy.org/`.

[76] Dan Jurafsky. Minimum edit distance - definition of minimum edit distance, 2014. `https://web.stanford.edu/class/cs124/lec/med.pdf`.

[77] Eran Kalige, Darrell Burkey, and IPS Director. A case study of eurograbber: How 36 million euros was stolen via malware. *Versafe/Check Point*, 2012.

[78] Anestis Karasaridis, Brian Rexroad, David A Hoeflin, et al. Wide-scale botnet detection and characterization. *HotBots*, 7:7–7, 2007.

[79] Ahmad Karim, Rosli Bin Salleh, Muhammad Shiraz, Syed Adeel Ali Shah, Irfan Awan, and Nor Badrul Anuar. Botnet detection techniques: review, future trends, and issues. *Journal of Zhejiang University SCIENCE C*, 15(11):943–983, 2014.

[80] Gordon Kelly. Report: 97% of mobile malware is on android. this is the easy way you stay safe, 2014. `http://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile-malware-is-on-android-this-is-the-easy-way-you-stay-safe/`.

[81] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 1951.

[82] lcamtuf. lcamtuf.coredump.cx. `http://lcamtuf.coredump.cx/`. [Last visited: 14-Jun-2017].

[83] Felix Leder and Tillmann Werner. Containing conficker, 2008.

[84] Felix Leder and Tillmann Werner. Know your enemy: Containing conficker. *The Honeynet Project, University of Bonn, Germany, Tech. Rep*, 2009.

[85] Robert Lemos. After rustock, botnet rebuilding underway, 2011. `http://www.csoonline.com/article/2128148/privacy/after-rustock--botnet-rebuilding-underway.html`.

[86] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[87] Chao Li, Wei Jiang, and Xin Zou. Botnet: Survey and case study. In *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*, pages 1184–1187. IEEE, 2009.

[88] Franklin Mark Liang. *Word Hy-phen-a-tion by Com-put-er*. PhD thesis, Citeseer, 1983.

[89] Hui-Tang Lin, Ying-You Lin, and Jui-Wei Chiang. Genetic-based real-time fast-flux service networks detection. *Computer Networks*, 57(2):501–513, 2013.

[90] Peng Liu, Wanyu Zang, and Meng Yu. Incentive-based modeling and inference of attacker intent, objectives, and strategies. *ACM Transactions on Information and System Security (TISSEC)*, 8(1):78–118, 2005.

[91] Chen Lu. *Network Traffic Analysis Using Stochastic Grammars*. PhD thesis, Clemson, SC, USA, 2012.

[92] Chen Lu and Richard Brooks. Botnet traffic detection using hidden markov models. In *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '11, pages 31:1–31:1, New York, NY, USA, 2011. ACM.

[93] Chen Lu, Jason M. Schwier, Ryan Craven, Lu Yu, Richard R. Brooks, and Christopher Griffin. A normalized statistical metric space for hidden markov models. *IEEE T. Cybernetics*, 43(3):806–819, 2013.

[94] Chen Lu, Jason M Schwier, Ryan M Craven, Lu Yu, Richard R Brooks, and Christopher Griffin. A normalized statistical metric space for hidden markov models. *IEEE transactions on cybernetics*, 43(3):806–819, 2013.

[95] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[96] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[97] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[98] Samuel Marchal, Jérôme François, Radu State, and Thomas Engel. Semantic based dns forensics. In *Information Forensics and Security (WIFS), 2012 IEEE International Workshop on*, pages 91–96. IEEE, 2012.

[99] Denis Maslennikov. Carberp-in-the-mobile, 2012. `http://www.securelist.com/en/blog/208194045/Carberp_in_the_Mobile`.

[100] Aleksandr Matrosov and Eugene Rodionov. Festi botnet analysis and investigation. 2011.

[101] Trend Micro. Global botnet threat activity map, 2016. `http://www.trendmicro.com/us/security-intelligence/current-threat-activity/global-botnet-map/index.html`.

[102] Microsoft. Microsoft works with financial services industry leaders, law enforcement and others to disrupt massive financial cybercrime ring, 2013. `http://blogs.technet.com/b/microsoft_blog/archive/2013/06/05/microsoft-works-with-financial-services-industry-leaders-law-enforcement-and-others-to-disrupt-massive-financial-cybercrime-ring/aspx`.

[103] Microsoft. Vpn tunneling protocols, 2015. `https://technet.microsoft.com/en-us/library/cc771298(v=ws.10).aspx`.

[104] Masnick Mike. China tries to block encrypted traffic. `https://www.techdirt.com/articles/20121217/10222821404/china-tries-to-block-encrypted-traffic.shtml`, 2011. [Last visited: 06-Aug-2015].

[105] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108. ACM, 2012.

[106] The Bro Network Security Monitor. Bro. `https://www.bro.org/`. [Last visited: 06-Aug-2015].

[107] Miranda Mowbray and Josiah Hagen. Finding domain-generation algorithms by looking at length distribution. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 395–400. IEEE, 2014.

[108] Julia Narvaez, Barbara Endicott-Popovsky, Christian Seifert, Chiraag Aval, and Deborah A Frincke. Drive-by-downloads. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10. IEEE, 2010.

[109] Jose Nazario. Blackenergy ddos bot analysis. *Arbor*, 2007.

[110] Jose Nazario. Bot and botnet taxonomy, 2008. http://www.slideshare.net/digitallibrary/bot-and-botnet-taxonomy.

[111] Jose Nazario and Thorsten Holz. As the net churns: Fast-flux botnet observations. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 24–31. IEEE, 2008.

[112] BBC News. Spam on rise after brief reprieve, 2008. `http://news.bbc.co.uk/2/hi/technology/7749835.stm`.

[113] NTOP project. ntopng: High-Speed Web-based Traffic Analysis and Flow Collection. `http://www.ntop.org/products/traffic-analysis/ntop/`. [Last visited: 14-Jun-2017].

[114] Jolie ODell. Bank login-stealing botnet found hiding in amazon cloud, 2009. http://readwrite.com/2009/12/10/zeus-botnet-amazon-cloud-ec2.

[115] Ilker Ozcelik, Yu Fu, and Richard R Brooks. Dos detection is easier now. In *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, pages 50–55. IEEE, 2013.

[116] Roberto Perdisci, Igino Corona, David Dagon, and Wenke Lee. Detecting malicious flux service networks through passive analysis of recursive dns traces. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 311–320. IEEE, 2009.

[117] Roberto Perdisci, Igino Corona, and Giorgio Giacinto. Early detection of malicious flux networks via large-scale passive dns traffic analysis. *Dependable and Secure Computing, IEEE Transactions on*, 9(5):714–726, 2012.

[118] Brian Prince. How pushdo malware hides c&c traffic. https://www.securityweek.com/how-pushdo-malware-hides-cc-traffic, 2013. [Last visited: 06-Aug-2015].

[119] The Tor Project. obfsproxy. https://www.torproject.org/projects/obfsproxy.html, 2015. [Last visited: 06-Aug-2015].

[120] The Tor project. The onion router (tor), 2016. https://www.torproject.org/.

[121] qpalzm01. Blackmail bitcoin ransomware (with sourcecode) plus free gift, pm me, 2017 (Last visited). http://pwoah7foa6au2pul.onion/listing.php?id=232704.

[122] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[123] NS Raghava, Divya Sahgal, and Seema Chandna. Classification of botnet detection based on botnet architechture. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, pages 569–572. IEEE, 2012.

[124] Jayaram Raghuram, David J Miller, and George Kesidis. Unsupervised, low latency anomaly detection of algorithmically generated domain names by generative probabilistic modeling. *Journal of Advanced Research*, 2014.

[125] Hal Roberts, Ethan Zuckerman, and John Palfrey. 2007 circumvention landscape report: Methods, uses, and tools. http://cyber.harvard.edu/sites/cyber.harvard.edu/files/2007_Circumvention_Landscape.pdf, 2007.

[126] RTPIS Lab. Real-Time Power and Intelligence Systems (RTPIS) Laboratory. http://rtpis.org. [Last visited: 06-Aug-2015].

[127] William Salusky and Robert Danford. Know your enemy: fast-flux service networks. *The Honeynet Project*, 2008. http://www.honeynet.org/papers/ff/.

[128] David Sancho. Steganography and malware: Concealing code and c&c traffic. http://blog.trendmicro.com/trendlabs-security-intelligence/steganography-and-malware-concealing-code-and-cc-traffic/, 2015. [Last visited: 06-Aug-2015].

[129] T. Savolainen, Nokia, J. Kato, NTT, T. Lemon, and Nominum Inc. Improved recursive dns server selection for multi-interfaced nodes, 2012. https://tools.ietf.org/html/rfc6731#page-24.

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

[131] Craig Schiller and James R Binkley. *Botnets: The killer web applications*. Syngress, 2011.

[132] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 101–102. IEEE, 2001.

[133] Jason Schwier. Pattern recognition for command and control data systems. 2009.

[134] Jason M Schwier, Richard R Brooks, Christopher Griffin, and S Bukkapatnam. Zero Knowledge Hidden Markov Model Inference. *Pattern Recognition Letters*, 30(14):1273–1280, 2009.

[135] Jason Montgomery Schwier. *Pattern Recognition for Command and Control Data Systems*. PhD thesis, Clemson, SC, USA, 2009. AAI3369281.

[136] SearchSecurity. Thinking fast-flux: New bait for advanced phishing tactics, 2013. http://searchsecurity.techtarget.com/tip/Thinking-fast-flux-New-bait-for-advanced-phishing-tactics.

[137] Panda Security. Virus encyclopedia: Machbot.a, 2008. http://www.pandasecurity.com/homeusers/security-info/197376/MachBot.A/.

[138] Panda Security. The cyber-crime black market:uncovered, 2011. http://press.pandasecurity.com/wp-content/uploads/2011/01/The-Cyber-Crime-Black-Market.pdf.

[139] SecurityFocus. Image attack on myspace boosts phishing exposure, 2007. http://www.securityfocus.com/brief/522.

[140] SecurityFocus. Imperfect storm aids spammers, 2007. http://www.securityfocus.com/news/11442.

[141] Cosma Rohilla Shalizi and James P Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *Journal of statistical physics*, 104(3-4):817–879, 2001.

[142] Cosma Rohilla Shalizi, Kristina Lisa Shalizi, and James P Crutchfield. Pattern discovery in time series, part i: Theory, algorithm, analysis, and convergence. *Journal of Machine Learning Research*, pages 02–10, 2002.

[143] Seungwon Shin, Guofei Gu, Narasimha Reddy, and Christopher P Lee. A large-scale empirical study of conficker. *Information Forensics and Security, IEEE Transactions on*, 7(2):676–690, 2012.

[144] shonajaan. Zeus botnet, 2017 (Last visited). `http://pwoah7foa6au2pul.onion/listing.php?id=8305`.

[145] Sérgio SC Silva, Rodrigo MP Silva, Raquel CG Pinto, and Ronaldo M Salles. Botnets: A survey. *Computer Networks*, 2012.

[146] Srgio S.C. Silva, Rodrigo M.P. Silva, Raquel C.G. Pinto, and Ronaldo M. Salles. Botnets: A survey. *Computer Networks*, 57(2):378 – 403, 2013. Botnet Activity: Analysis, Detection and Shutdown.

[147] Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science*, 24(4):265–269, 1973.

[148] Sourcefire. Snort. `https://www.snort.org/`. [Last visited: 06-Aug-2015].

[149] Paul Sroufe, Santi Phithakkitnukoon, Ram Dantu, and João Cangussu. Email shape analysis for spam botnet detection. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–2. IEEE, 2009.

[150] ICANN SSAC. Sac 025 ssac advisory on fast flux hosting and dns : Fast and double flux attacks. 2008.

[151] Matija Stevanovic and Jens Myrup Pedersen. An efficient flow-based botnet detection using supervised machine learning. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 797–801. IEEE, 2014.

[152] Alastair Stevenson. Hackers turning to tor network to hide evolved malware, warns kaspersky lab. `http://www.v3.co.uk/v3-uk/news/2335401/hackers-turning-to-tor-network-to-hide-evolved-malware-warns-kaspersky-lab`, 2014. [Last visited: 06-Aug-2015].

[153] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM.

[154] Zhan Su, Byung-Ryul Ahn, Ki-Yol Eom, Min-Koo Kang, Jin-Pyung Kim, and Moon-Kyun Kim. Plagiarism detection using the levenshtein distance and smith-waterman algorithm. In *Proceedings of the 2008 3rd International Conference on Innovative Computing Information and Control*, ICICIC '08, pages 569–, Washington, DC, USA, 2008. IEEE Computer Society.

[155] Symantec. The sality botnet, 2010. http://www.symantec.com/connect/blogs/sality-botnet.

[156] Symantec. Spyeye bot versus zeus bot, 2010. `http://www.symantec.com/connect/blogs/spyeye-bot-versus-zeus-bot`.

[157] Symantec. Trojan.zeroaccess, 2011. `http://www.symantec.com/security_response/writeup.jsp?docid=2011-071314-0410-99`.

[158] Yuta Takata, Shegeki Goto, and Tatsuya Mori. Analysis of redirection caused by web-based malware. *Proceedings of the Asia-Pacific Advanced Network*, 32:53–62, 2011.

[159] Rapid7 team. Welcome to project sonar!, 2013. `https://community.rapid7.com/community/infosec/sonar/blog/2013/09/26/welcome-to-project-sonar`.

[160] Team-Cymru. A taste of http botnets. 2008.

[161] Jason Thatcher. Personal communication. Unpublished, 2014.

[162] TheRegister. Storm botnet blows itself out, but will the zombie network rise again?, 2008. http://www.theregister.co.uk/2008/10/14/storm_worm_botnet_rip/.

[163] Jignesh Vania, Arvind Meniya, and HB Jethva. A review on botnet and detection technique. *International Journal of Computer Trends and Technology*, 4(1):23–29, 2013.

[164] SophosLabs Vanja Svajcer, Principal Researcher. Sophos mobile security threat report, 2014. `http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-mobile-security-threat-report.pdf`.

[165] Bart Vanluyten, Jan C Willems, and Bart De Moor. Equivalence of state representations for hidden markov models. *Systems & Control Letters*, 57(5):410–419, 2008.

[166] var462. [account stealer pack] [crack pack] [cracking tools [2016] + free gift, 2017 (Last visited). `http://pwoah7foa6au2pul.onion/listing.php?id=134707`.

[167] vimproducts. 24/7 layer 7 ddos http/website (rent 25k botnet) (flat rate & guaranteed downtime), 2017 (Last visited). `http://pwoah7foa6au2pul.onion/listing.php?id=133745`.

[168] John Von Neumann and Oskar Morgenstern. Theory of games and economic behavior. *Bull. Amer. Math. Soc*, 51(7):498–504, 1945.

[169] Kuochen Wang, Chun-Ying Huang, Shang-Jyh Lin, and Ying-Dar Lin. A fuzzy pattern-based filtering algorithm for botnet detection. *Computer Networks*, 55(15):3275–3286, 2011.

[170] Ping Wang, Sherri Sparks, and Cliff Changchun Zou. An advanced hybrid peer-to-peer botnet. *Dependable and Secure Computing, IEEE Transactions on*, 7(2):113–127, 2010.

[171] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 109–120. ACM, 2012.

[172] Wikipedia. Receiver operating characteristic, 2016. `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`.

[173] Wikipedia. Reverse dns lookup, 2016. `https://en.wikipedia.org/wiki/Reverse_DNS_lookup`.

[174] Philipp Winter and Stefan Lindskog. How china is blocking tor. *arXiv preprint arXiv:1204.0447*, 2012.

[175] Philipp Winter, Tobias Pulls, and Juergen Fuss. Scramblesuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 213–224. ACM, 2013.

[176] Wireshark. tshark - Dump and analyze network traffic. `https://www.wireshark.org/docs/man-pages/tshark.html`. [Last visited: 14-Jun-2017].

[177] Julia Wolf. Technical details of srizbis domain generation algorithm, 2008.

[178] James Wyke. Was microsoft's takedown of citadel effective?, 2013. `http://nakedsecurity.sophos.com/2013/06/12/microsoft-citadel-takedown/`.

[179] Mengjun Xie, Heng Yin, and Haining Wang. Thwarting e-mail spam laundering. *ACM Transactions on Information and System Security (TISSEC)*, 12(2):13, 2008.

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

[181] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated domain-flux attacks with dns traffic analysis. *Networking, IEEE/ACM Transactions on*, 20(5):1663–1677, 2012.

[182] Kyle York. Dyn statement on 10/21/2016 ddos attack, 2016. `http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/`.

[183] Lu Yu. *Stochastic Tools for Network Security: Anonymity Protocol Analysis and Network Intrusion Detection*. PhD thesis, Clemson University, 2012.

[184] Lu Yu, Jason M Schwier, Ryan M Craven, Richard R Brooks, and Christopher Griffin. Inferring statistically significant hidden markov models. *IEEE Transactions on Knowledge and Data Engineering*, 25(7):1548–1558, 2013.

[185] Xiaocong Yu, Xiaomei Dong, Ge Yu, Yuhai Qin, Dejun Yue, and Yan Zhao. Online botnet detection based on incremental discrete fourier transform. *Journal of Networks*, 5(5):568–576, 2010.

[186] Xiaonan Zang, Athichart Tangpong, George Kesidis, and David J Miller. Botnet detection through fine flow classification. *unpublished, Departments of CS&E and EE, The Pennsylvania State University, University Park, PA, Report No. CSE11-001*, 2011.

[187] Hossein Rouhani Zeidanloo, Azizah Bt Abdul Manaf, Rabiah Bt Ahmad, Mazdak Zamani, and Saman Shojae Chaeikar. A proposed framework for p2p botnet detection. *International Journal of Engineering and Technology*, 2(2):161, 2010.

[188] Hossein Rouhani Zeidanloo, Mohammad Jorjor Zadeh Shooshtari, Payam Vahdani Amoli, M Safari, and Mazdak Zamani. A taxonomy of botnet detection techniques. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 2, pages 158–162. IEEE, 2010.

[189] Han Zhang, Manaf Gharaibeh, Spiros Thanasoulas, and Christos Papadopoulos. Botdigger: Detecting dga bots in a single network. 2016.

[190] Junjie Zhang, Xiapu Luo, Roberto Perdisci, Guofei Gu, Wenke Lee, and Nick Feamster. Boosting the scalability of botnet detection using adaptive traffic sampling. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 124–134. ACM, 2011.

[191] Yao Zhao, Yinglian Xie, Fang Yu, Qifa Ke, Yuan Yu, Yan Chen, and Eliot Gillum. Botgraph: Large scale spamming botnet detection. In *NSDI*, volume 9, pages 321–334, 2009.

[192] Xingsi Zhong, Afshin Ahmadi, Richard Brooks, Ganesh Kumar Venayagamoorthy, Lu Yu, and Yu Fu. Side channel analysis of multiple pmu data in electric power systems. In *Power Systems Conference (PSC), 2015 Clemson University*, pages 1–6. IEEE, 2015.

[193] Xingsi Zhong, Paranietharan Arunagirinathan, Afshin Ahmadi, Richard Brooks, Ganesh Kumar Venayagamoorthy, Lu Yu, and Yu Fu. Side Channel Analysis of Multiple PMU Data in Electric Power Systems. In *Power System Conference (PSC), 2015 Clemson University*, pages 1–6, March 2015.

[194] Xingsi Zhong, Yu Fu, Lu Yu, Richard Brooks, and G. Kumar Venayagamoorthy. Stealthy malware traffic not as innocent as it looks. In *Malicious and Unwanted Software: The Americas (MALWARE), 2015 10th International Conference on*, October 2015.

[195] Zhaosheng Zhu, Guohan Lu, Yan Chen, Zhi Judy Fu, Phil Roberts, and Keesook Han. Botnet research survey. In *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, pages 967–972. IEEE, 2008.