DNS Traffic Analysis for Network-based Malware Detection

LINH VU HONG

KTH Information and
Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

KTH Royal Institute of Technology
School of Information and Communication Technology
Master's Programme in Security and Mobile Computing - NordSecMob

DNS Traffic Analysis for Network-based Malware Detection

Linh Vu Hong
linhvh@kth.se

Master's Thesis
Stockholm - April 16, 2012

Home supervisor: Prof. Gerald Q. Maguire Jr. (*maguire@kth.se*)
KTH Royal Institute of Technology, Sweden

Host supervisor: Assoc. Prof. Christian W. Probst (*probst@imm.dtu.dk*)
Technical University of Denmark, Denmark

Industrial advisor: Michael Liljenstam, PhD (*michael.liljenstam@ericsson.com*)
Ericsson Security Research, Sweden

Abstract

(English)

Botnets are generally recognized as one of the most challenging threats on the Internet today. Botnets have been involved in many attacks targeting multinational organizations and even nationwide internet services. As more effective detection and mitigation approaches are proposed by security researchers, botnet developers are employing new techniques for evasion. It is not surprising that the Domain Name System (DNS) is abused by botnets for the purposes of evasion, because of the important role of DNS in the operation of the Internet. DNS provides a flexible mapping between domain names and IP addresses, thus botnets can exploit this dynamic mapping to mask the location of botnet controllers. Domain-flux and fast-flux (also known as IP-flux) are two emerging techniques which aim at exhausting the tracking and blacklisting effort of botnet defenders by rapidly changing the domain names or their associated IP addresses that are used by the botnet. In this thesis, we employ passive DNS analysis to develop an anomaly-based technique for detecting the presence of a domain-flux or fast-flux botnet in a network. To do this, we construct a lookup graph and a failure graph from captured DNS traffic and decompose these graphs into clusters which have a strong correlation between their domains, hosts, and IP addresses. DNS related features are extracted for each cluster and used as input to a classification module to identify the presence of a domain-flux or fast-flux botnet in the network. The experimental evaluation on captured traffic traces verified that the proposed technique successfully detected domain-flux botnets in the traces. The proposed technique complements other techniques for detecting botnets through traffic analysis.

Sammanfattning

(Svenska)

Botnets betraktas som ett av de svåraste Internet-hoten idag. Botnets har använts vid många attacker mot multinationella organisationer och även nationella myndigheters och andra nationella Internet-tjänster. Allt eftersom mer effektiva detekterings - och skyddstekniker tas fram av säkerhetsforskare, har utvecklarna av botnets tagit fram nya tekniker för att undvika upptäckt. Därför är det inte förvånande att domännamnssystemet (Domain Name System, DNS) missbrukas av botnets för att undvika upptäckt, på grund av den viktiga roll domännamnssystemet har för Internets funktion - DNS ger en flexibel bindning mellan domännamn och IP-adresser. Domain-flux och fast-flux (även kallat IP-flux) är två relativt nya tekniker som används för att undvika spårning och svartlistning av IP-adresser av botnet-skyddsmekanismer genom att snabbt förändra bindningen mellan namn och IP-adresser som används av botnets. I denna rapport används passiv DNS-analys för att utveckla en anomali-baserad teknik för detektering av botnets som använder sig av domain-flux eller fast-flux. Tekniken baseras på skapandet av en uppslagnings-graf och en fel-graf från insamlad DNS-trafik och bryter ned dessa grafer i kluster som har stark korrelation mellan de ingående domänerna, maskinerna, och IP-adresserna. DNS-relaterade egenskaper extraheras för varje kluster och används som indata till en klassificeringsmodul för identifiering av domain-flux och fast-flux botnets i nätet. Utvärdering av metoden genom experiment på insamlade trafikspår visar att den föreslagna tekniken lyckas upptäcka domain-flux botnets i trafiken. Genom att fokusera på DNS-information kompletterar den föreslagna tekniken andra tekniker för detektering av botnets genom trafikanalys.

Preface

This thesis was prepared at Ericsson Security Research, Ericsson AB, Sweden in partial fulfillment of requirements for the M.Sc. degree in Security and Mobile Computing (NordSecMob) from KTH Royal Institute of Technology in Sweden and Technical University of Denmark (DTU) in Denmark.

The thesis deals with anomaly based techniques for detecting malware based on network traffic analysis. The main focus is on detecting fast-flux and domain-flux botnets based on the analysis of Domain Name System (DNS) traffic.

The thesis consists of a summary report and a prototype system written in Python 2.7 and MATLAB® 2010b to demonstrate the proposed technique.

Stockholm, April 2012

Linh Vu Hong

Acknowledgements

I would like to give my gratitude to Prof. Gerald Q. Maguire Jr., my home university supervisor, for guiding me all the way through my thesis. His immense knowledge and experience in the field have helped me with all the obstacles that I encountered during my thesis work. He always finds time to help students on all subjects that matter despite his busy schedule. It was my pleasure to have him as my supervisor for my thesis.

I thank my host university supervisor, Prof. Christian W. Probst, who gave me invaluable suggestions and comments regarding my thesis.

I am grateful to Dr. Michael Liljenstam, my industrial advisor, for always being available for all the questions that I came up with during my thesis. His knowledge and industrial experience have helped me to produce fruitful result in this thesis project.

I would also like to thank Prajwol Kumar Nakarmi and András Méhes, my colleagues at Ericsson Security Research, for helping me during the experimental phase.

I am thankful to Ericsson Security Research, for providing me the opportunity and necessary facilities to conduct my thesis project. I would like to thank all my colleagues at Ericsson Security Research for a wonderful, friendly, and intellectual experience.

I want to express my love for my family, friends, and Vera.

Stockholm, April 2012

Linh Vu Hong

Acronyms

Notation	Description
ASN	Autonomous System Number
C&C	Command and Control
CDN	Content Distribution Network
DFA	Deterministic Finite Automaton
DGA	Domain Generation Algorithm
DNS	Domain Name System
FFSN	Fast-Flux Service Network
GPRS	General Packet Radio Service
HTTP	HyperText Transfer Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
IRC	Internet Relay Chat
ISP	Internet Service Provider
NIDS	Network-based Intrusion Detection System
NMTF	Nonnegative Matrix Tri-Factorization
P2P	Peer-to-Peer
TLD	Top Level Domain Name
URL	Uniform Resource Locator

Glossary

Notation	**Description**
bot	a malicious program that infects and recruits the host to join the botnet
botmaster	the entity that controls the botnet
botnet	a network of compromised computers controlled by a botmaster
DGA	an algorithm to automatically generate domain names that are pseudo-random. Domain-flux botnets often use such an algorithm to generate their domain names
DNS failure graph	a bipartite graph represents the mapping between domain names and the IP addresses of host interfaces that generated a query, this graph is constructed from the information extracted from failed DNS queries
DNS lookup graph	a bipartite graph represents the mapping between fully qualified domain names and the IP addresses mapped to them, this graph is constructed from the information extracted from successful DNS queries
DNS	a hierarchical, distributed naming system used to map from domain names to their corresponding IP addresses *and* for other mappings
domain-flux	a technique based upon changing the domain name very frequently with (usually) algorithmically generated domain names
fast-flux	a technique for changing the IP addresses associated with a domain at high frequency
URL-flux	a technique using a username generation algorithm (so the URLs are associated with different user profiles) to disguise the command and control traffic in botnet

Contents

Abstract i

Sammanfattning ii

Preface iii

Acknowledgements iv

Acronyms v

Glossary vi

1 Introduction 1
- 1.1 Problem Statement and Methodology . . . 1
- 1.2 Scope of the Thesis . . . 2
- 1.3 Structure of the Thesis . . . 2

2 Background 3
- 2.1 Domain Name System . . . 3
- 2.2 Botnet . . . 5
  - 2.2.1 Structure of a Botnet . . . 5
  - 2.2.2 Operation of a Botnet . . . 7
  - 2.2.3 Command and Control Model . . . 8
  - 2.2.4 Command and Control Channel Evasion . . . 10
- 2.3 Fast-flux and Domain-flux . . . 11
  - 2.3.1 Fast-flux . . . 11
  - 2.3.2 Domain-flux . . . 14
  - 2.3.3 URL Flux . . . 16
- 2.4 Intrusion Detection Systems in Fixed and Mobile Networks . . . 17
  - 2.4.1 Intrusion Detection and Preventation Systems . . . 17
  - 2.4.2 Network-based Intrusion Detection Systems . . . 19
  - 2.4.3 NIDS in Mobile Networks . . . 19
- 2.5 Related Work . . . 20

3 DNS Traffic Analysis 24
- 3.1 DNS Graph . . . 25
  - 3.1.1 DNS Lookup Graph . . . 25
  - 3.1.2 DNS Failure Graph . . . 25
  - 3.1.3 Community Structure of DNS Graph . . . 26
- 3.2 Analysis Procedure . . . 28
- 3.3 Graph Construction Module . . . 30
  - 3.3.1 DNS Data Extraction . . . 30
  - 3.3.2 Anonymization . . . 30
  - 3.3.3 Data Filtering . . . 31
  - 3.3.4 Graph Construction . . . 33
- 3.4 Graph Decomposition Module . . . 33

3.4.1 Graph Decomposition 34
3.4.2 The NMTF Co-clustering Method 35
3.4.3 Interpretation of NMTF Results 36
3.4.4 Obtaining Coherent Co-clusters 38
3.4.5 Practical Issues 39
3.5 Feature Extraction Module 39
3.6 Regression Function Module 43

4 Results and Discussion 45
4.1 Prototype Implementation 45
4.2 Data Collection 46
4.2.1 Traffic Data 46
4.2.2 Data Labeling and Filtering 46
4.3 Experimental Scenarios 48
4.3.1 Offline Scenario 48
4.3.2 Online Scenario 49
4.4 Goodness Metrics 49
4.5 Results 50
4.5.1 Offline Scenario 53
4.5.2 Online Scenario 55
4.6 Discussion 57

5 Conclusions and Future Work 59

Bibliography 61

A Parameters for Analysis Procedure 66

Chapter 1

Introduction

Currently internet security is facing an undergoing transformation and evolution of threats. More sophisticated techniques are being used by attackers, especially in attacks targeting multinational organizations or nationwide internet services. The motivation for these attacks are not limited to financial gain, but frequently include political and ideological purposes. In many attacks, a *botnet* is utilized to increase the number of hosts involved in the attack. Such attacks are one of the challenging threats to internet security today.

1.1 Problem Statement and Methodology

Many methods have been used to detect the presence of a botnet in a network by analyzing traffic in order to discover correlated activities of hosts in the network. However, the problem with such methods is that sometimes it is not possible to analyze the traffic, for example when the traffic is encrypted. As botnets increasingly use fast-flux and domain-flux, the tasks of analyzing traffic becomes more complicated, especially when the amount of traffic in the network is large.

In this project we proposed a technique of analyzing the Domain Name System (DNS) [1, 2] traffic in a network to detect the presence of a botnet in the network. The proposed technique is an anomaly-based technique, and it can be integrated into a Network-based Intrusion Detection System (NIDS)[3] to detect incidents in a network.

DNS is widely used by botnets to locate the Command and Control (C&C) servers and the DNS traffic is always available[*]. Thus, DNS analysis is promising method to detect the presence of a botnet, especially a botnet which uses fast-flux or domain-flux, two emerging botnet evasion techniques based on DNS abuse. Of course, DNS traffic analysis alone can not provide highly accurate detection results. However, DNS can provide complementary evidence that can be combined with the results of other traffic analysis techniques to increase the accuracy of botnet detection - as compared to DNS based analysis alone or to other traffic analysis. More concretely we claim that using the proposed DNS traffic analysis increases the *specificity* of the detection of fast-flux and domain-flux botnets.

[*] Except for the case of DNSCrypt and DNSSEC which are not (yet) widely deployed, thus we opted not to consider these types of DNS traffic in our work.

1.2 Scope of the Thesis

This thesis focuses on detecting the presence of a botnet in a network by examining the DNS traffic. This thesis does not consider botnets that do not use DNS as a method to locate their C&C server(s). Furthermore, in this thesis, encrypted and authenticated DNS traffic such as DNSCrypt [4] or DNSSEC [5] traffic are not considered[†].

1.3 Structure of the Thesis

The rest of this thesis is divided into four chapters. Chapter 2 starts by describing the background for this thesis including fundamentals of botnets and Network-base Intrusion Detection Systems (NIDS). Chapter 3 describes in detail of our proposed approach of inspecting the DNS traffic to detect the presence of a botnet exhibiting fast-flux and/or domain-flux behaviors. Afterwards, Chapter 4 summarizes our experimental results and discusses the success or failure of our proposed technique. Chapter 5 concludes the thesis and suggests some future work.

[†] DNSCrypt was developed by OpenDNS in order to encrypt regular DNS traffic, while DNSSEC is a suite of IETF's specifications to provide authenticity and integrity for DNS, but not availability or confidentiality.

Chapter 2

Background

This chapter gives a brief introduction to the Domain Name System, botnets, and Intrusion Detection Systems. Additionally, fast-flux and domain-flux, two of the the emerging botnet detection evasion techniques, are also described. Readers who are familiar with these topics can skim or skip corresponding sections.

2.1 Domain Name System

Domain Name System (DNS) is a hierarchical, distributed naming system that provides a critical Internet service of mapping between two principal namespaces on the Internet: domain name hierarchy and Internet Protocol (IP) address space [1, 2]. By translating the human-friendly (i.e. easy for human to remember) domain names into IP addresses, DNS makes it possible to assign domain names to a group of Internet resources in a meaningful way and independent of entities' physical location(s). This naming mechanism keeps the names of the Internet resources remain consistent even if there are changes in the IP addresses of underlying networks. This is advantageous for the users, machines, and services because they can cite Internet resources in the meaningful way, but without having to worry how these resources are actually located.

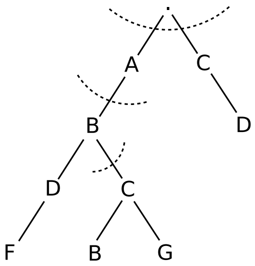


Figure 2.1: The DNS Tree [6]

The domain name space is structured in hierarchical manner as a tree. Each domain name consists of multiple domain name labels separated by a ".". A domain name identifies a path from the root node of the DNS hierarchy, denoted by the rightmost ".", to a node representing the domain name. This node contains a set of resource information associated with the domain name in the form of a collection of resource records (RRs). For example a domain name `F.D.B.A.` is a path from root node to node `F`. This node in the DNS tree contains information about the domain name `F.D.B.A.` as shown in Figure 2.1. The depth of the node in a DNS tree is called the *domain level*, for example, `A.` is a Top Level Domain Name (TLD), `B.A.` is second level domain, and so on.

The DNS system is split into multiple *zones* [1] by partitioning the domain name space between sibling nodes. Each zone is responsible for a group of nodes, hence their corresponding domain names and associated information for that zone is *authoritative*. The zone is identified by the domain name of the node closest to the root node. Each zone has one or more *authoritative name server(s)* where RRs of its domain names are stored. These authoritative name servers have complete knowledge of the domain names within their zone and serve this information when requested. The authoritative name servers can further delegate their authority over part of the zone to other authoritative name servers. This hierarchical model makes DNS the largest distributed system on the Internet. This model and DNS's caching mechanism provides fault-tolerance ability for the DNS system.

The domain name resolution process is depicted in Figure 2.2. A client can resolve a domain name `www.example.com` using a *stub resolver* that is built-in to all systems that have an Internet connection. This stub resolver sends a DNS query request to a *recursive DNS resolver* (RDNS). If RDNS has information about the queried domain name cached, then this information will be returned to the stub resolver. Otherwise, the RDNS starts by querying the root name server, then the root name server redirect the RDNS to the name server that is authoritative for the TLD "`com.`". This process continues until the RDNS reaches a name server that is authoritative for `www.example.com`. It will query this name server for relevant RRs and return a response to the stub resolver on the client. The RDNS also caches these RRs for the domain name `www.example.com` for a certain period of time so that it can immediately respond if there is other request for these RRs for this domain name. The period of time that the RRs are cached in RDNS depends on the Time To Live (TTL) value contained in the information returned for RRs associated with the domain name `www.example.com` from the authoritative name server.

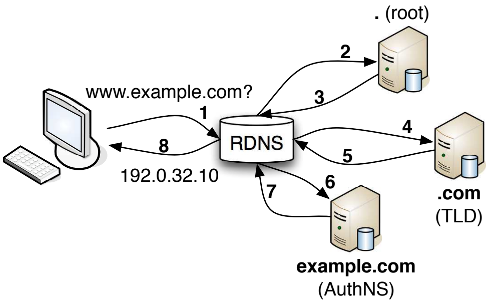


Figure 2.2: The DNS resolution process [6]

2.2 Botnet

A *botnet* is a network of compromised computers [7, 8], called *bots* or *zombies*, which are the under control of a *botmaster*. These computers, mainly personal computers, are exploited and infected without the user's knowledge by a malicious self-propagated program called a *bot*. After successfully infecting a host, the new bot reports its existence to a C&C server in order to join the botnet and then awaits instructions from this botmaster. The bot may also start to collect sensitive information such as keystrokes, contact lists, passwords, or financial information from the compromised host. The information collected is sent to the C&C server. The population of a botnet gradually increases as the botnet incorporates new victims. Botnets with thousands or even millions of bots have been observed in the wild. The *botmaster* via the C&C server instructs the botnet to conduct various malicious activities which may include stealing sensitive information, spreading malware via massive spamming, performing Distributed Denial of Services attack (DDoS), generating advertisements to commit fraud, conducting click fraud, or phishing. With a large population and a huge volume of traffic, botnets can target large multinational companies or even nationwide internet services. The distributed, dynamic, and anonymous nature of botnets makes it difficult to detect and defend against them. The fact that there are more and more attacks involving botnets confirms the danger of botnets and reenforces their roles in cybercrime. Understanding the structure and operation of botnets are key factors to effectively detect and mitigate them. The following sections provide some background knowledge about botnets and their current patterns of evolution.

2.2.1 Structure of a Botnet

Regardless of architecture, size (in number of nodes), and communication protocol, all botnets consist of four components: the botmaster, the bots, the C&C server(s), and the communication channels between them. The basic structure is shown in the Figure 2.3 and described in the following paragraphs. In practice, botnets can optionally have one or more layers of proxies between the C&C server(s) and the bots.

- **Botmaster:** The botmaster is an entity that controls the operation of botnet, and decides how the botnet brings benefits to her by performing attacks, or the botmaster may rent their botnet to others. The botmaster decides what operation(s) the botnet should perform and when the botmaster's commands are executed by the botnet, i.e., which instructions are sent to the bots.
- **Bots:** Each bot is a piece of software which combines the behaviors of previously known malware, especially the features of a worm or trojan horse. A computer infected with such a program is turned into a zombie and is recruited to the botnet. The term *bot* is used to indicate both the malicious

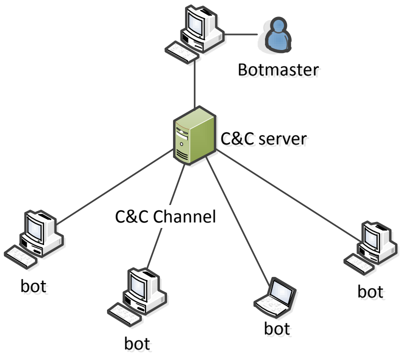


Figure 2.3: Botnet structure

program itself and the computer infected by such a program. The infection vectors are various, for example the bot could be installed in the victim's computer by exploiting the system's vulnerabilities or by convincing a user to execute a program attached to an email message. The bots are the army which directly conducts malicious activities under the control of the botmaster. The number of bots continuously increases as each bot tries to infect other computers. For example, each bot may incorporate other computers in the same physical network or obtain the user's contact list and send malware attached to email messages to all of the user's contacts. The population of a botnet is one measure of its danger. Upon instruction by the botmaster or periodically, each bot downloads and updates itself to a new version which is more resilient, offers new evasion techniques, and/or new exploits and attack methods. In this manner the botnet can continue to evolve. It has been observed that some botnets implement an affiliate program in which the bot includes and operates additional malware within its "shell"(see for example the TLD-4 bot [9]).

- **C&C servers:** The C&C servers are intermediates through which the botmaster controls the botnet and rallies the bots to perform an attack. C&C servers may also be used to host malicious software to be downloaded by the bots. Where C&C servers are placed and how they are configured depend on the C&C model of the botnet. In a centralized model, the C&C servers are often hosted in hostile sites or a bullet-proof network (also known as *bullet-proof hosting*, a service provided by some domain hosting or web hosting firms that allow their customers to bypass the laws or contractual terms of service regulating Internet content and service use); while in a peer-to-peer model, these servers are usually selected among those bots with public accessibility, high bandwidth connectivity, and high availability. A variety of communication protocols can be used by the botnet. For example if the Internet Relay Chat (IRC) protocol is used, then the C&C server can be installed as an IRC server; while if the botnet uses the HyperText Transfer Protocol (HTTP) protocol then the C&C server can be a web server. C&C servers are the only component of the botnet which communicates with the botmaster (potentially via one or more layers of proxies in order to circumvent any tracing effort by the authorities). Thus, many botnet's evasion techniques include hiding the identity of the C&C servers in order to prevent authorities from easily shutting down the botnet or taking control of the botnet.

- **C&C channel:** The C&C channel utilizes some communication protocols to distribute the instructions from the C&C servers to the bots. As this coordination empowers the botnet, the communication channel plays the vital role in the existence of the botnet and the effectiveness of the botnet's attacks. A botnet's communication channel can either be a push or pull channel: in the push channel, the bots wait for the C&C servers to actively contact them with instructions; while in the pull channel, the bots periodically contact one of the C&C servers for instructions. The encrypted IRC protocol was widely used by early botnets for their communication. This protocol is still used by many current botnets. However, as IRC becomes a less common protocol and as defenders pay increasing attention to detection of botnets based upon their communication signatures botnets are shifting to more sophisticated and agile communication methods. For example, the communication can be disguised as common and legitimate network traffic using HTTP or Peer-to-Peer (P2P) protocols. Additionally, some botnets hide instructions in postings on social network sites. There is also increasing use of public key cryptography to authenticate the C&C servers and their instructions, thus making it hard for others to take control of a given botnet.

- **Proxies (optional):** Since the C&C servers are the gateways through which the botmaster controls the botnet, the botnet can be mitigated by identifying and shutting down the C&C servers which the bots communicate with. Therefore, adding one or more layer(s) of proxies between the bots and C&C servers helps to hide the identities of the C&C servers. The presence of proxies in a botnet's structure is optional in the sense that these proxies are not required for the botnet's operation, but they are useful to evade detection and mitigation.

Each of the above components plays a different role in the botnet. The C&C servers and the communication channels are considered the most important components and also the weakest links of the botnet. Therefore, botnets are often classified by their communication channels and most of the defenses against botnets are aimed at detecting and shutting down the C&C servers or breaking the communication channels. In reality, depending on the purpose and evasion strategy of the bot's authors, there could be additional components introduced into a botnet.

2.2.2 Operation of a Botnet

It is essential to understand how botnets operate in order to implement an effective defense mechanism against them. Figure 2.4 illustrates a common botnet's operation. There are five specific phases that are of specific concern to us in the operation of a botnet. These phases are described below.

(1) In the beginning, suitable victims are infected with malicious code. This code is often obfuscated by encryption in order to evade signature-based Intrusion Detection System (IDS). Upon execution in the victim computer, the bot de-obfuscates itself and performed a number of system calls to disguise itself (for example by modifying the file system or attaching itself to legitimate system services). The bot in the infected computer begins to harvest sensitive information such as keystrokes or the user's credentials. A comprehensive case study of *Rustock* [10], the world-biggest botnet which was taken down in March 2011 by coordinated effort of both ISPs and software vendors, provides further details of this phase of a botnet's operation.

(2) In this second phase the bot contacts the C&C servers to join the botnet.

(3) When the bot receives commands from the C&C server it sends back the collected information, downloads the latest update, or infects new victims.

(4) The bot finds new victims to infect by exploring the local network or send the malware attached to email to all of the people in the user's contact list. The botnet's population is gradually growing and the botnet is expanding across networks.

(5) Given a sufficient number of bots, the botmaster can instruct the botnet to conduct various malicious activities. This is when the botnet potentially brings profits to the botmaster. We note that the botnet continues expanding while it carries out these malicious activities. In practice the botmaster uses the botnet both to increase the size of the botnet itself [11] and to carry out malicious activities.

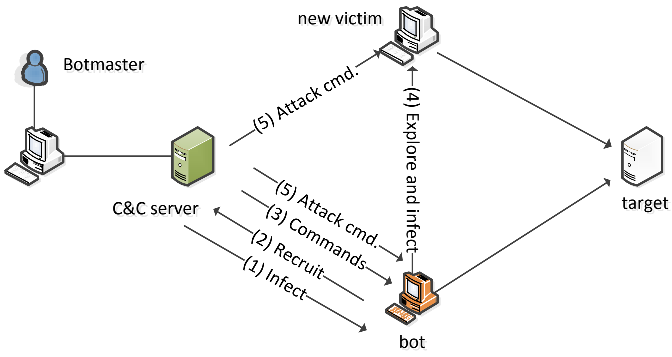


Figure 2.4: Basic operation of a botnet

2.2.3 Command and Control Model

The communication between bots and C&C servers plays a vital role in maintaining a botnet's existence. Botnets are useless, isolated compromised computers without of this communication because the bots cannot join the bots army or receive instructions from the botmaster to coordinate their malicious activities. An exception to this is the case of a “fire-and-forget” type of botnet that has a single purpose and does not receive updates or further commands. Note that such a botnet’s attack has to be pre-configured and controlled by other sources - for example in the case of a fictional example via news [12]. This C&C communication is unique to each botnet and it probably changes between different variants of a given botnet. Therefore it is an important characteristic to categorize botnets into different classes. According to their topologies, botnet C&Cs models can be roughly divided into three types of models: centralized, Peer-to-Peer (P2P), and unstructured C&C models.

Centralized Model

In the centralized model, there are one or a few C&C servers which act as rendezvous points for the botnet. Usually, these servers are hosted in some *bullet-proof* hosting service providers or in hostile sites, thus they are difficult to take down. In the first generation of botnets, these C&C servers were IRC servers with encrypted private channels to communicate with the botmaster. The bots join the private IRC channel and listen for instructions. The centralized model is very effective in terms of coordination since it allows the botmaster to easily control thousands or millions of bots and also guarantees low latency message delivery. On the other hand, this model has some drawbacks, such as the C&C servers are a single point of failure, hence if they are detected and taken down then the botnet ceases to operate. Despite these drawbacks, the centralized C&C model is still the most prevalent C&C model in practice because this model is simple to implement and easy to customize via a rich variety of support tools. A variant of the centralized model is the hierarchical model in which the bots are divided into different classes with a hierarchical relationship between them. This variant is more scalable and is more resilient to failure and mitigation.

Peer-to-Peer Model

When multiple organizations, such as Internet Service Providers (ISPs) or software vendors, started to coordinate their efforts to detect and mitigate botnets, some botnets circumvented this coordinated effort by shifting to a Peer-to-Peer (P2P) model. Figure 2.5 illustrates this P2P model for C&C of a botnet.

In contrast to the centralized model, in P2P model, there is no centralized C&C server, the botmaster can contact and send instructions to any peer (bot). The communication in this model does not depend on one or a few selected C&C servers, thus the botnet becomes more resilient to detection and mitigation. As a result, the defenders can not defeat this type of botnet by taking down a small number of bots. Nevertheless, implementing a P2P C&C model requires more work by the bot’s authors because of constraints posed by the P2P model.

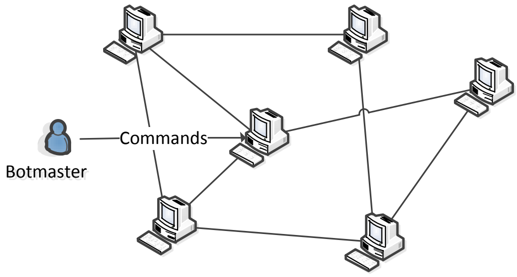


Figure 2.5: Peer-to-Peer Command and Control Model

For example, the underlying P2P model of communications does not guarantee message delivery, the latency in a P2P network is unpredictable, and a good management mechanism is required to control the large number of bots in the botnet in order to prevent bot enumeration and injection. The *Storm* worm [13] and *Nugache* are some examples of botnets using this P2P model. The increase in the number of botnets using P2P model is evidence that bot authors are finding appropriate solutions to these constraints. Their effort has been motivated by the requirement for resilience becoming more vital for their botnets.

Unstructured Model

The bots in an unstructured C&C model do not contact and report to the C&C server. When the botmaster wants to start an attack, he or she scans the Internet to learn of available bots and sends instructions to them. This kind of botnet is the most difficult to discover and take down. Although we have not yet seen any botnets of this kind in the wild the current rapid evolution of botnets makes it necessary to study the behaviors of such botnets and figure out effective countermeasures for this C&C model.

2.2.4 Command and Control Channel Evasion

It is vital for the botnet to hide its communication channel and the identities of the C&C servers and botmaster. Hence, many techniques have been employed by bot authors in order to evade the security counter-measures that have been deployed.

Use of an encrypted communication channel between the C&C servers and bots is one of the first evasion methods that was utilized in order to avoid being detected by signature-based IDS. As noted earlier, these bots used an encrypted IRC channel as their communication channel since IRC is very easy to implement and deploy and it allows the C&C server to control thousands of bots with very low latency. However, today IRC traffic now can be easily examined by an IDS, especially note that it is uncommon to use IRC in the network. For this reason botnets are shifting to new approaches to disguise their C&C traffic as common application traffic, such as HTTP or P2P traffic. Recently bots are hiding their communication messages in social network posts, news feeds, or using steganography. There are observations of bots using cryptography to authenticate the C&C servers and their instructions.

The techniques used by botnets to hide the identities of the C&C server(s) are evolving over time. Initially, botnets used a hard-coded list of C&C server IP addresses to locate their C&C servers. As this hard-coded technique became less effective and these IP addresses were easily blacklisted, more sophisticated techniques are employed by botnets providing a stealthier way to locate their C&C servers. An emerging technique is to use the DNS to provide botnets with a flexible way of contacting their C&C servers. In the next section, fast-flux and domain-flux, two DNS-based techniques that have recently begun to be used by botnets are described in detail.

2.3 Fast-flux and Domain-flux

As blacklisting techniques used by defenders to disrupt malicious activities became more effective, more sophisticated techniques were employed by bot authors in order to evade this detection and mitigation effort. DNS is an essential protocol on the Internet as it provides a convenient mapping between domain names and their corresponding IP addresses. Ironically, the flexibility and global availability of DNS is now being abused by botnets to build their resilient command and control infrastructure. Fast-flux and domain-flux are two techniques which are now being widely integrated into botnets in order to enhance their resilience.

These are not new techniques, as the principles underlying these techniques have been used by legitimate services for a long time. Details of these techniques are described in the following subsections. Their usage in malicious activities is evidence of the evolution of techniques used by attackers to create more robust botnets. This thesis will focus on the approaches to detect botnets which employ fast-flux and domain-flux.

2.3.1 Fast-flux

Fast-flux (also known as IP-flux) is described by ICANN as [14]:

> *"rapid and repeated changes to A and/or NS resource records in a DNS zone, which have the effect of rapidly changing the location (IP address) to which the domain name of an Internet host (A) or name server (NS) resolves"*

Technically, this rapid change is done by setting a short Time To Live (TTL) for a given DNS record and changing the associated IP addresses frequently. This technique has been used for a long time as a load balancing method by legitimate systems, e.g. heavy loaded and highly available websites, such as internet search engines or a Content Distribution Network (CDN).

As cybercrime evolved, this technique is now used in a malicious way by forming a Fast-Flux Service Network (FFSN), a network of compromised hosts which share the same domain name(s). The IP addresses of hosts are rapidly swapped in and out of the DNS entry based on the hosts' availability and bandwidth. Usually FFSNs employ a load distribution scheme by appointing some hosts to check the health of other nodes in the network. With such rapid change of the IP addresses in FFSNs, it is nearly impossible to disrupt the FFSNs by simply blocking specific IP addresses. A possible solution is to suspend the domain names used by FFSNs. However, this is a very tedious task since these domain names are usually registered at registrars who are resistant to blocking of domain names.

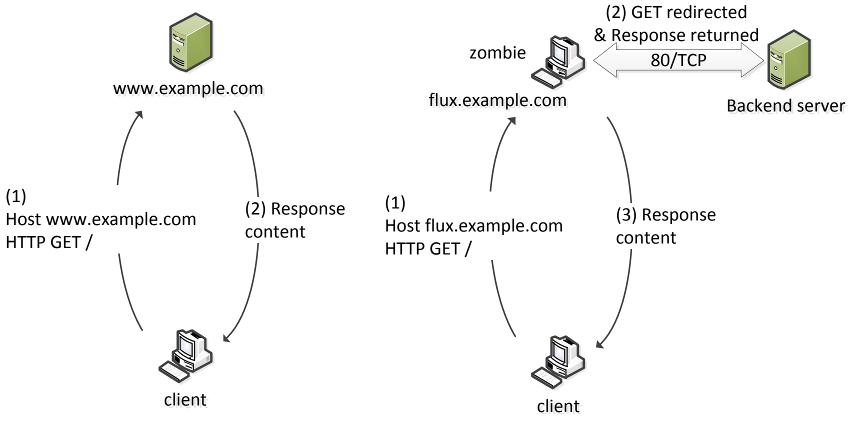
Another vantage of FFSNs is their high level of anonymity as the compromised hosts are usually set up as *blind proxy redirectors* which funnel the traffic to and from the backend servers which serve the actual, malicious content. By funneling the traffic, the blind proxy redirectors disrupt any attempt to track and reveal the identities of the backend servers. Consequently, this increases the life time of the backend servers and makes it simpler to set up and manage the backend servers. Even if we can locate these backend servers, it is difficult to take them down as they normally reside in complicit or hostile networks. Due to the high level of anonymity and resilience, FFSNs have been observed participating in many different types of malicious practices, such as illegal content hosting, malware distribution, online pharmacy shops, or phishing. FFSNs are normally part of the botnet and the nodes selected to be in the FFSN are selected from compromised hosts with a globally accessible IP address and high bandwidth. FFSNs hide the C&C channel and host the malicious code of the bot.

There are two types of FFSN: single-flux and double-flux. In the former, only the host records are changed while in the latter, both host and name server resources records are changed. Figure 2.6 illustrates the life cycle of a web request in a single-flux network compared to that of a normal server. The difference is although the users believe that they are browsing content from the legitimate server, the FFSN redirector has redirected the request from the proxy to the backend server and redirects the response from this backend server to the users.

Listing 2.1 shows results of `dig`[*] command for a single-flux service network. We can see that in the set of IP addresses returned after 30 minutes for the same domain name, there are two new IP addresses. Furthermore, most of the IP addresses in the returned IP set are located in home networks.

In the double-flux service network, when the user requests content one more step

[*] `dig` (domain information groper) is a network administration command-line tool for querying DNS name servers. `Dig` is a built-in tool in almost all Linux distributions.



(a) Normal network

(b) Single-flux network

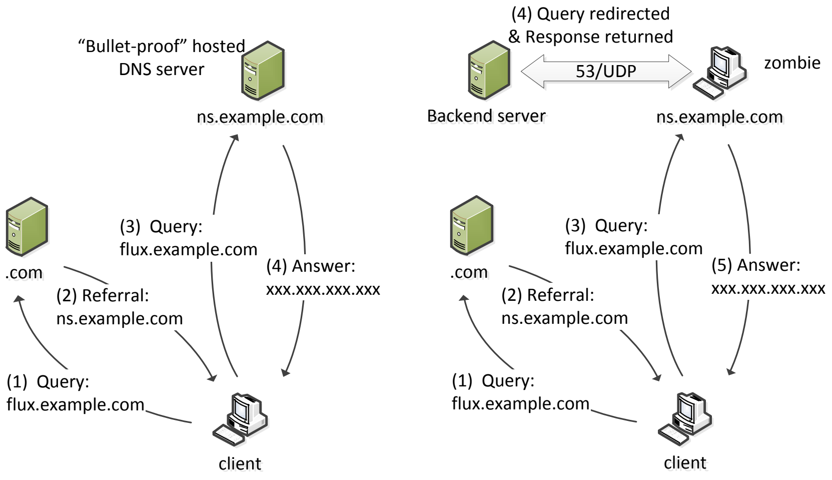
Figure 2.6: Web request to a single-flux service network compared to request to normal network (adapted from [15])

Listing 2.1: Results of `dig` command for a single-flux domain name [15]

```
 ;; WHEN: Sat Feb 3 20:08:08 2007
divewithsharks.hk. 1800 IN A 70.68.187.xxx [xxx.vf.shawcable.net]
divewithsharks.hk. 1800 IN A 76.209.81.xxx [SBIS-AS - AT&T Internet Services]
divewithsharks.hk. 1800 IN A 85.207.74.xxx [adsl-ustixxx-74-207-85.bluetone.cz]
divewithsharks.hk. 1800 IN A 90.144.43.xxx [d90-144-43-xxx.cust.tele2.fr]
divewithsharks.hk. 1800 IN A 142.165.41.xxx [142-165-41-xxx.msjw.hsdb.sasknet.sk
    .ca]

divewithsharks.hk. 1800 IN NS ns1.world-wr.com.
divewithsharks.hk. 1800 IN NS ns2.world-wr.com.

ns1.world-wr.com.  87169 IN A 66.232.119.212 [HVC-AS - HIVELOCITY VENTURES CORP]
ns2.world-wr.com.  87177 IN A 209.88.199.xxx [vpdn-dsl209-88-199-xxx.alami.net]

;; WHEN: Sat Feb 3 20:40:04 2007 (~30 minutes/1800 seconds later)
divewithsharks.hk. 1800 IN A 24.85.102.xxx [xxx.vs.shawcable.net] NEW
divewithsharks.hk. 1800 IN A 69.47.177.xxx [d47-69-xxx-177.try.wideopenwest.com]
     NEW
divewithsharks.hk. 1800 IN A 70.68.187.xxx [xxx.vf.shawcable.net]
divewithsharks.hk. 1800 IN A 90.144.43.xxx [d90-144-43-xxx.cust.tele2.fr]
divewithsharks.hk. 1800 IN A 142.165.41.xxx [142-165-41-xxx.msjw.hsdb.sasknet.sk
    .ca]

divewithsharks.hk. 1800 IN NS ns1.world-wr.com.
divewithsharks.hk. 1800 IN NS ns2.world-wr.com.

ns1.world-wr.com.  85248 IN A 66.232.119.xxx [HVC-AS - HIVELOCITY VENTURES CORP]
ns2.world-wr.com.  82991 IN A 209.88.199.xxx [vpdn-dsl209-88-199-xxx.alami.net]
```

takes place in the process to resolve the authoritative name servers for the domain. In a double-flux network, the name servers are compromised hosts that redirect the DNS queries to a backend DNS server and return the responses from this server to the user's client. Figure 2.7 depicts the difference between single-flux and double-flux while resolving the domain name.

Listing 2.2 illustrates results of `dig` command for a domain-flux service network.



(a) Single-flux (b) Double-flux

Figure 2.7: Double-flux domain resolution compared to single-flux (adapted from [15])

In this example, not only the IP addresses returned for the A records are changed, but also those for NS records. Note that similar to the single-flux service network, the set of IP addresses returned are mostly located in an *xDSL*[†] or dial up network. That means the interfaces associated with these IP addresses are mostly personal computers located in people's homes.

2.3.2 Domain-flux

Domain-flux is another approach to frustrate the defenders' efforts to block access to backend or C&C servers. As opposed to fast-flux in which the IP addresses are rapidly swapped, in domain-flux the domain names are changed instead. A huge number of domains names (or subdomains) are hard-coded into the bot's executable or are generated by an algorithm embedded in the bot called a *Domain Generation Algorithm (DGA)*. The botmaster has the same list of domain names, but only registers one or a few of them. These domain names point to the C&C servers. The bots find the address the C&C server by sequently or randomly resolving domain names in the list until it successfully locates a C&C server at the domain name registered by the botmaster. An example of domain names generated by *Kraken* [16, 17] bot is shown in Listing 2.3.

Botnets using domain-flux vary widely in number of generated domain names, the algorithm used, and how the algorithm is seeded. For example, the *Conficker-A* bot uses the current date and time as a seed to generate 250 domain names every three hours. This number is increased to 50 thousand domain names in

[†] xDSL standard of ADSL, SDSL, VDSL, etc. – all of these are types of digital subscriber line modems used to connected a customer premises model with an access network operator's packet data network over the twisted copper pairs used for analog telephony.

Listing 2.2: Results of `dig` command for a double-flux domain name [15]

```
;; WHEN: Wed Apr 4 18:47:50 2007
login.mylspacee.com. 177 IN A 66.229.133.xxx [c-66-229-133-xxx.hsd1.fl.comcast.
    net]
login.mylspacee.com. 177 IN A 67.10.117.xxx [cpe-67-10-117-xxx.gt.res.rr.com]
login.mylspacee.com. 177 IN A 70.244.2.xxx [adsl-70-244-2-xxx.dsl.hrlntx.swbell.
    net]

mylspacee.com. 108877 IN NS ns3.myheroisyourslove.hk.
mylspacee.com. 108877 IN NS ns4.myheroisyourslove.hk.
mylspacee.com. 108877 IN NS ns5.myheroisyourslove.hk.

ns1.myheroisyourslove.hk.854 IN A 70.227.218.xxx [ppp-70-227-218-xxx.dsl.sfldmi.
    ameritech.net]
ns2.myheroisyourslove.hk.854 IN A 70.136.16.xxx [adsl-70-136-16-xxx.dsl.bumttx.
    sbcglobal.net]
ns3.myheroisyourslove.hk. 854 IN A 68.59.76.xxx [c-68-59-76-xxx.hsd1.al.comcast.
    net]

;; WHEN: Wed Apr 4 18:51:56 2007 (~4 minutes/186 seconds later)
login.mylspacee.com. 161 IN A 74.131.218.xxx [74-131-218-xxx.dhcp.insightbb.com]
     NEW
login.mylspacee.com. 161 IN A 24.174.195.xxx [cpe-24-174-195-xxx.elp.res.rr.com]
     NEW
login.mylspacee.com. 161 IN A 65.65.182.xxx [adsl-65-65-182-xxx.dsl.hstntx.
    swbell.net] NEW

 ;; WHEN: Wed Apr 4 21:13:14 2007 (~90 minutes/4878 seconds later)
ns1.myheroisyourslove.hk. 3596 IN A 75.67.15.xxx [c-75-67-15-xxx.hsd1.ma.comcast
    .net] NEW
ns2.myheroisyourslove.hk. 3596 IN A 75.22.239.xxx [adsl-75-22-239-xxx.dsl.chcgil
    .sbcglobal.net] NEW
ns3.myheroisyourslove.hk. 3596 IN A 75.33.248.xxx [adsl-75-33-248-xxx.dsl.chcgil
    .sbcglobal.net] NEW
```

Listing 2.3: Domain names generated by Kraken

```
rbqdxflojkj.mooo.com           fvkvwf.dynserv.com
bltjhzqp.dyndns.org            duxhvrrb.mooo.com
cffxugijxn.yi.org              natiouwx.dyndns.org
etllejr.dynserv.com            afmbtgyktty.yi.org
ejfjyd.mooo.com                eltxytxurng.dynserv.com
mnkzof.dyndns.org              tmuncana.mooo.com
adhbtib.yi.org                 wafyfrryfzr.dyndns.org
vqsqul.dynserv.com             lbimniu.yi.org
dawjjopw.mooo.com              fqjhsvsevdy.dynserv.com
jamptmlvrw.dyndns.org          zstyderw.mooo.com
ihouxyds.yi.org                ouwyrav.dyndns.org
```

the *Confiker-C* bot. Another example is the *Torpig* bot which seeds the DGA with one of the most popular trend topics in Twitter [18]. Nevertheless, all of these bots have some characteristics or requirements in common. Firstly, the number of domain names generated by bots has to be large enough in order to successfully exhaust a blacklisting effort. Secondly, the generated domains should not collide with existing domain names. These characteristics of domain flux suggest some defenses. For instance, as the number of domain names being generated increases, before reaching a domain name that is actually registered by the botmaster, the bots cause a significant number of failed DNS queries in the network when they query for the nonexistent domain names. This could be considered as abnormal behavior in the network, thus suggesting there is domain-flux [19]. In order to avoid collisions with existing domains, the generated domain names usually appear to be random strings, i.e., they are *unpronounceable* usually have no meaning, unlike legitimate services in which domain names usually consist of meaningful words from a dictionary and they are *pronounceable*, thus this characteristic might indicate domain-flux. Some bots try to overcome this problem by using algorithms which generate domains from English (or other language) words. An example is the algorithm used by *Kraken* bot, which combines the generated words with common English word suffixes such as -*able*, -*dom*, or -*ly*. The phonetic features of these words are also exploited by some algorithms to make the generated words more likely to be pronounceable.

Domain-flux serves the botmaster very well, as the botmaster only needs to register a few domain names in the list in order to keep the botnet running, while the defenders have to register or block all the domain names in the list to stop the botnet's operation. Furthermore, in the case of algorithmically generated domain names finding the algorithm requires reverse engineering the bot to know the list of domain names that can be generated, and this reverse engineering task takes quite a long time.

A technique similar to domain-flux is also used by some social networks or blog services such as *Wordpress*[‡] or *Tumblr*[§] to customize the user's pages by modifying the subdomain of the Uniform Resource Locator (URL). However, most of the domain names generated by such services are associated with one or few base domains and they are more persistent as compared to the domain names used by domain-flux. The key difference is that the legitimate services generate many subdomains of a small number of domains, while the botnets try to spread their usage over a much larger part of the whole domain name space.

In the wild, fast-flux and domain-flux can be combined together to provide a high level of anonymity, resilience, and availability for the C&C servers.

2.3.3 URL Flux

A similar but more economical approach than domain-flux is URL-flux which was introduced in the Android bot by Xiang [20]. Instead of using domain names, URL-flux uses Web 2.0 sites as a mean of communication. Similarly to domain-flux, the bot in URL-flux has a list of usernames generated by a *Username Generation Algorithm (UGA)* from which it selects a username to visit on a Web 2.0 site. If the user exists, then the bot uses a hard-coded public key to validate the most recent messages for this user. If this validation succeeds, then the message will be considered as instructions issued by the botmaster. With URL-flux, bots disguise their C&C traffic as normal HTTP traffic, making it difficult to detect the bot's activities in the network. As Web 2.0 is becoming increasingly popular,

[‡] http://wordpress.com

[§] http://www.tumblr.com

further investigations should be conducted in order to figure out how to detect this C&C communication and defend against this type of evasion. However, a common practice in Web 2.0 sites is to locate a user's page based on the relative path of the URL which is not related to DNS traffic. This investigation is out of the scope of this thesis, thus we do not consider URL-flux further.

2.4 Intrusion Detection Systems in Fixed and Mobile Networks

As mentioned in Section 1.1, our proposed technique is developed based on the principles of an Intrusion Detection System (IDS). Our proposed technique uses an anomaly-based approach to detect the presence of a botnet in a network, and it can be integrated in a NIDS to detect incidents in a network.

This section begins by describing what an IDS is and how it can be used to prevent intrusions from having a large negative impact on the network and the hosts. Next, NIDS - a type of IDS based on monitoring network traffic is presented. Finally, we discuss the use of NIDS in mobile networks.

2.4.1 Intrusion Detection and Preventation Systems

Intrusion detection is defined by National Institute of Standards and Technology (NIST) as [3]:

> *"the process of monitoring the events occurring in a computer system or network and analyzing them for the signs of possible **incidents**, which are violations of imminent threats of violation of computer security policies, acceptable use policies, or standard security practices."*

An Intrusion Detection System (IDS) is a software system which automates the intrusion detection process. An IDS seeks to detect incidents in the system in advance of the occurrence of a violation of some policy. There are many causes of incidents in the system, such as malware infection, an attacker gaining access to the system or probing system for vulnerabilities, or misuses of the system by authorized users resulting in a violation of security policies. The goal of the IDS is to alert the system's administrators about such incidents in their early stage *before* they cause any damage and to support the system's administrators in their response to malicious incidents. Although many incidents are malicious in nature, others are not, for instance an user might mistype an address resulting in an attempt to access a critical system to which he or she is unauthorized. Therefore an IDS should be able to classify potentially malicious incidents with reasonable accuracy, i.e., a low rate of *false negatives* and *false positives.* A false negative occurs when a malicious incident is classified as innocent and a false positive occurs when an innocent incident is classified as malicious.

An Intrusion Prevention System (IPS) is a system which has all the functionality of an IDS but also can make some attempt(s) to prevent the incident from occurring [3]. The extend of the actions taken by an IPS in response to incidents depends on how aggressive the system is and how the IPS is configured by the system's administrators. In practice, the prevention features of IPS are usually disabled and the system works simply as an IDS because the attempts made in response to incidents may result in inconvenience for users or interrupt the services.

A typical IDS uses many approaches to detect incidents in the systems. The NIST guide to intrusion detection systems [3] describes three common approaches: signature-based, anomaly-based, and stateful protocol analysis.

- **Signature-based:** In this approach, incidents are detected by comparing code or a data fragments to a set of signatures (i.e., patterns) of known threats. This method is very effective at detecting known threats, however it is ineffective at detecting unknown threats or new variants of known threats. Furthermore, a signature-based IDS has to maintain a huge database of signatures and the recognition process involves searching for matches over a collection of thousands or even millions of signatures. Thus, an efficient searching algorithm is very important for signature-based detection method.

- **Anomaly-based:** Unlike the signature-based approach, in an anomaly-based method the actual behavior is compared to a collection of pre-defined "normal" behaviors. If the deviation from normal behavior is out of acceptable range, then the system will raise an alarm. An advantage of anomaly-based IDS over signature-based IDS is that an anomaly-based system can detect unknown threats. However, it may produce a high rate of false positives, for example when a new service is added to the system. Furthermore, it takes a period of time to build a profile of the "normal" behaviors of the system, this period of time is called the *training period.* The normal behavior profile can either be static or dynamic. A static profile is unchanged during the operation period of the IDS until a new "normal" profile is regenerated. In contrast, a dynamic profile is gradually updated with observed data during operation. A static profile might be outdated overtime, while a dynamic profile might produce inaccurate detection results because of faulty learning.

- **Stateful protocol analysis:** Stateful protocol analysis detects malicious activities by comparing the actual behavior to vendor-developed profiles which specify how a particular protocol should or should not behave. It provides the ability to track the state of a stateful protocol which allows it to detect some attacks that other methods cannot detect. However, creating a universal profile (model) of a protocol is a hard task and sometimes it is nearly impossible. An additional drawback of this method is that it cannot detect attacks which are disguised as normal behaviors of the protocol.

In practice, an IDS often uses multiple methods to provide broader and more accurate detection results.

2.4.2 Network-based Intrusion Detection Systems

A NIDS monitors the network traffic for particular network segments or devices and analyzes the network, transport, and application protocols to identify suspicious activities [3]. NIDS are widely deployed by many organizations at their network's boundary. The sensor which collects network data for the NIDS can be deployed in two modes: in inline mode, the sensor is placed so that all the traffic goes through it; while in passive mode, the sensor receives a copy of the network traffic (thus avoiding increasing the delay experienced by the network traffic).

NIDS provides various ranges of security capabilities. At a minimum it collects information about the hosts in the network, such as which operating system is being run on each host and applications. An NIDS generally has extensive logging and data capturing ability. Most NIDS use multiple detection methods ranging from signature-based to in-depth stateful protocol analysis of common protocols. Some NIDS are able to overcome the evasion techniques used by malware, hence improving the accuracy of the detection results.

There are limitations associated with NIDS as well. Since NIDS monitors all the traffic in the network, their performance are critical especially in a large network with high load. In reality, NIDS often employ multiple layers of filters to reduce the amount of traffic going through the expensive analysis process. This also helps to increase the accuracy of the detection, as it reduces the noise in the input data. Another disadvantage of NIDS is that it is susceptible to some attacks involving a large amount of traffic, such as Denial of Services Attack. Thus, the ability to resist such kinds of attacks is an important consideration when designing an NIDS.

2.4.3 NIDS in Mobile Networks

Rapid development of mobile phone industry has led to mobile devices and equipment with extensive computing capability and connectivity comparable to those of fixed computers. This growth has led to more attention both from attackers and security researchers. The ability to download and install third-party applications in almost all smart-phones increases the probability that malicious code will be installed in a mobile phone without the user's knowledge. As the mobile network now is becoming an all-IP network, it is exposed to many internet threats including botnets. Therefore it is necessary to integrate an IDS into the mobile network's architecture to detect and defend against these traditional internet threats.

2.5 Related Work

DNS analysis is an emerging technique for detecting malicious activities in the network. It has been claimed by many researchers that DNS analysis is an effective technique for botnets detection, especially those with fast-flux and domain-flux behaviors.

As one of the first efforts, T. Holz et al. presented the characteristics of FFSN and introduced a technique to detect a FFSN in a network [21]. Observing the behaviors of the FFSN, Holz et al. proposed a weighted linear regression for assigning a *flux-score* for a domain name,i.e., a score that determines how likely it is that a domain belongs to a FFSN. The regression function is a linear function of three features: number of unique A records, number of distinct Autonomous System Numbers (ASNs), and number of unique NS records. However, the weights in the Holz's regression function depend on the characteristics of the network in which the system is deployed.

R. Perdisci et al. proposed a technique to detect *in-the-wild* FFSNs by analyzing the passive DNS traces [22] in three steps: first reduce the amount of traffic using filter rules, then cluster domains into groups using a similarity measure that is a Jacard Index that compares the set of resolved IP addresses between two domain names, and finally apply a statistical supervised learning approaches to a number of passive and active features to determine if a domain belongs to a FFSN. Perdisci et al. also discussed how spam detection can be benefit from this technique.

For detecting domain-flux, S. Yadav et al. grouped domain names in the DNS queries together by their top level domains (TLDs), mapped IP address set, or the connected component in a bipartite IP-domain graph [18]. The technique is based on the observation that algorithmically generated domain names have a different distribution of their alphanumeric characters or bigrams compared to normal domain names due to the randomness in the generated domain names. They have proposed metrics to capture the characteristics of the algorithmically generated domain names, specifically they propose to use Kullback-Leibler divergence on the distribution of unigrams and bigrams of group of domain names, a Jaccard index that compares the set of bigrams between a benign domain and a malicious one, and computing the edit distance to convert a domain name to another in order to capture the randomness of a group of domain names. Additionally, Yadav et al. claimed that component analysis of the IP-domain mapping graph increases the detection rate.

Y. Jin et al. studied *network-wide communication patterns* by constructing a bipartite graph of traffic flows between hosts inside a network and hosts outside, a so called *traffic activity graph* (TAG) [23]. The authors observed that the graph of network traffic has a community structure which reflects the social relationships of these applications' users, i.e., the network traffic graph consists of multiple strongly connected components which are isolated from others. Among those strongly connected components, there are some giant components which can be further divided into smaller components which are loosely connected to each other by a small number of edges. In other words, the giant components consist of a number of dense subgraphs which show intensive activities between their nodes. Decomposing such giant connected components using the co-clustering algorithm *tri-Nonnegative Matrix Factorization* (also known as Nonnegative Matrix Tri-Factorization or Orthogonal Nonnegative Matrix Tri-Factorization) [24], the authors discovered that each application exhibits a different communication pattern that is represented by different structures of components in the traffic flow graph. These structures of components can be categorized into four types that represent different interactions between hosts inside and outside of a network. This information can be used to detect malicious activities in a network. However, Jin et al. did not provide any method to use the component structure of the graph for malware detection. Furthermore, the decomposition algorithm tNMF has high complexity, and its optimal parameters depend heavily on the characteristics of the network. Jin et al. only provided a method to statically find the optimal parameters, even though a dynamic method would be desired in practice [23].

N. Jiang et al. applied the Jin et al. findings to detect suspicious activities in a network through DNS failure graph analysis [19]. Jiang et al. constructed a bipartite graph for the failed (or *unproductive*) DNS queries, based on the the intuition that a computer usually is infected by more than one instance of malicious software, and thus they exhibit a correlated communication pattern in their DNS queries. The *tri-Nonnegative Matrix Factorization* (also known as Nonnegative Matrix Tri-Factorization or Orthogonal Nonnegative Matrix Tri-Factorization) algorithm is used to extract the dense subgraphs from the DNS failure graph. Afterwards, these subgraphs are categorized and their evolution tracked over time. Based on their evolution, they are able to determine if hosts and domains in a subgraph are malicious [19]. However, in their work, Jiang et al. did not point out the specific method for using this information to detect malicious activities in a network.

As DNS analysis approach gained more attention from the security research community, more systematic approaches were developed to utilize DNS to detect malicious activities. Two domain name reputation systems have been proposed: Notos by Antonakakis et al. [25] and EXPOSURE by L. Bilge et al. [26]. These are two large scale passive DNS analysis systems for detecting malicious activities that have been built and deployed in large ISP's network. The premise of these system is that malicious use of DNS service has unique characteristics which can be use to distinguish malicious use from benign use. Thus, in these two systems, their respective authors built a model of normal DNS usage based on a set of features which were extracted from captured DNS traffic. The deviation of malicious domain names's behaviors are identified by a machine learning approach for anomaly detection.

In Notos [25], the statistical features of a domain name is divided into three different planes: network-based features, zone-based features, and evidence-based features. While network-based features are used to rule out the benign domain names which have stable DNS usage, the zone-based features are extracted from the domain names themselves (by string analysis or `whois` information) and are employed to distinguish malicious domain names and legitimate domain names that belong to known CDNs. In the evidence-based features plane, the domain names and their mapped IP addresses are checked against known malicious domain names and IP addresses. The reputation score of a domain name is derived by a Decision Tree using Logit-Boost strategy (LAD) reputation function.

Similarly, EXPOSURE [26] uses a set of 15 features that are divided into four groups: time-based features, DNS answer-based features, TTL value-based features, and domain name-based features. These features aim to detect different aspects of the malicious domain names behaviors. The time-based features capture the short-lived domain names which have regularly repeating patterns by using the cumulative sum change point detection algorithm. The DNS answer-based features expose domain names whose mapped IP addresses spread over multiple autonomous systems (ASs). The TTL value-based features detect domain names that have short TTL values and if those values are changed frequently. Finally, the domain name-based features identify domain names that would be "strange looking" to users. The J48 decision tree algorithm is used for the classifier to determine whether a domain name is malicious based on its features [26].

Continuing from previous work on building Notos, a domain name reputation system [25], Antonakakis et al. proposed Kopis, a system for monitoring a overview at the high level of the DNS hierarchy in order to discover the anomaly in malicious DNS activities [6]. Unlike previous detection systems such as Notos [25] or EXPOSURE [26], that rely on passive monitoring of recursive DNS (RDNS) traffic at a limited number of RDNS servers, Kopis takes advantages of the global visibility of DNS traffic at the upper levels of the DNS hierarchy to detect malicious domain names. The random forest classifier is used as the machine learning algorithm to build a model for the detector from a number of statistical features that represent the resolution patterns of legitimate and malicious domain names.

M. Knysz [27] discussed various forms of mimicry attacks against current FFSN detection systems. These attacks exploit the fact that almost all current FFSN detection systems are based on the distinction between the behaviors fo the FFSN and benign domain names. With the leverage of accurate models for bot decay, online availability, DNS advertisement, and performance, the botmaster can construct novel mimicry attacks that can bypass current FFSN detection systems such as the ones proposed by Holz. et al. [21], Passerini et al. [28], Hu et al. [29], Huang et al. [30], and Perdisci et al. [22]. These mimicry attacks are increasing adopted by FFSNs to circumvent simple and fast detection systems, although botmasters have not managed to produce the accurate models assumed by this study yet. Nevertheless, it is necessary to develop more sophisticated and advanced techniques for detecting FFSN.

Hu et al. [31] provides an interesting measurement and analysis at a global scope of

IP-usage patterns of fast-flux botnets. They deployed a system of 240 lightweight DNS probing engines spread over multiple geographical locations all over the world to collect DNS data of both legitimate and malicious domain names. This data gives an interesting global view of IP-usage patterns, such as the number of IP addresses employed, the overlap between the IP addresses of the A records and NS records, and how the IP address set mapped to a domain name grows. Their insight view provides some interesting features that could be used to detect FFSN botnets.

Hao et al. [32] study the initial query patterns of malicious domain names during short period after they are registered. This study revealed a number of interesting findings related to the initial behaviors of malicious domain names, including the time between registration and attack, the location of the DNS infrastructure, and their early lookup behaviors. These findings could be used as a guide for design and building early warning systems for detecting malicious domain names based on DNS analysis.

Chapter 3

DNS Traffic Analysis

As discussed in Chapter 2, the operation of a botnet which employs fast-flux and/or domain-flux depends heavily on the use of DNS. Although bots have behaviors similar to legitimate services, fast-flux or domain-flux botnets have some characteristics due to their nature. For example, FFSNs are mostly composed of bots running on personal hosts, thus the botmaster cannot choose nodes with particular IP addresses; while the IP addresses of legitimate services are carefully chosen by the service providers to meet their customers's requirements. One of the important customer requirements is low delay, hence the service should be located “near” in the network to the user. In contrast, the botnet may utilize non-local communication to communicate between the local bots and the C&C servers. This suggests that there should be a graphical clustering of IP addresses for legitimate services and a diversity of IP addresses for the C&C servers. Furthermore, there is no guarantee of the uptime of the bots because the botmaster does not have physical control of the hosts. Thus, the hosts in FFSN can come and go any time, much like the nodes in a P2P network of similar hosts [21]. Or in domain-flux botnets, in order to exhaust the blacklisting effort, a lot of domain names are generated and most of them are unregistered domain names. Additionally, as noted in Chapter 2 the queries from bots for the generated (or listed) domain names of C&C servers will result in multiple failed DNS queries to a large set of domain names within a short time window which rarely happens for legitimate services.

A common practice in defending against botnets is to detect and mitigate their communication infrastructure by inspecting the network traffic and discovering abnormal patterns such as IRC traffic or the signature of the botnet's commands. However, it is not always feasible to access the data at the application layer as these data may be encrypted. Furthermore, inspecting traffic to and from an individual endpoint will not uncover the correlation between endpoints that is a strong evidence of the presence of a botnet in a network, especially botnets that employ fast-flux and domain-flux.

In this thesis, we proposed to construct an anomaly-based botnet detection technique based on DNS traffic analysis. The main idea of our approach is to build a graph from the DNS traffic and then analyze this graph to discover distinctive features of malicious activities. The DNS graph allows us to observe the correlation between different endpoints in the network. Furthermore, DNS traffic is always available* and the amount of DNS traffic is rather small compared to other traffic in the network. Although DNS traffic analysis alone cannot

* As stated in the scope of the thesis, we did not consider DNSCrypt or DNSSEC traffic.

produce highly accurate detection results, it can provide complementary evidence to determine the presence of a botnet in a network. An appropriate combination with other traffic analysis techniques is needed to provide high accuracy botnet detection.

3.1 DNS Graph

Inspired by the work of Y. Jin et al. [23], we constructed two graphs from captured DNS traffic: a DNS lookup graph and DNS failure graph. These two graphs are expected to capture the correlation between domain names and IP addresses (or hosts) in a network. This correlation will be used to detect the presence of fast-flux and domain-flux network in the traffic. As we will see later the former type of graph helps to detect fast-flux, while the later helps to detect domain-flux. The DNS lookup graph and DNS failure graph are formally defined in the following subsections. The DNS lookup graph and DNS failure graph are constructed for each observation period T (also called an *epoch*), often an epoch of one day ($T = 1\, day$) is used to maximize the number of correlations observed and eliminate the effect of IP address churn [33]. Although we only considered IP version 4 (IPv4) [34] in our work, the application of our approach for IP version 6 (IPv6) [35] is analogous.

3.1.1 DNS Lookup Graph

For each *epoch* T of DNS traffic, we will use $\mathcal{D}$ to denote the set of unique domain names queried by hosts in the network and $\mathcal{I}$ to denote the set of unique IP addresses returned for those domain names. A *DNS lookup graph* is a weighted bipartite graph $\mathcal{G}_L := \{\mathcal{D} \times \mathcal{I}, \mathcal{E}\}$ where an edge $e = (d, i, w) \in \mathcal{E}$ exists between the domain name $d \in \mathcal{D}$ and the IP address $i \in \mathcal{I}$ if i is returned for d during the observation epoch and the weight w indicates how many times i is returned for d in this set of queries. Intuitively, the DNS lookup graph shows the mapping between domain names and their corresponding IP addresses. Note that the DNS lookup graph is built only from the successful DNS queries, i.e, those with a response code of `NO_ERROR` [1, 2].

3.1.2 DNS Failure Graph

Similar to the DNS lookup graph, the DNS failure graph is built from the failed DNS queries which have a response code other than `NO_ERROR` [1, 2] in the captured network traffic during a *epoch* T. Let $\mathcal{D}$ denote the set of unique domain names appearing in failed DNS queries and $\mathcal{H}$ be the set of hosts issuing those failed queries. A *DNS failure graph* is a weighted bipartite graph $\mathcal{G}_F := \{\mathcal{D}\times\mathcal{H}, \mathcal{E}\}$ where an edge $e = (d, h, w) \in \mathcal{E}$ exists between a domain name $d \in \mathcal{D}$ and a host

$h \in \mathcal{H}$ if h issues at least one failed query for d during the observation period and the weight w indicates the number of failed queries for d originating from h.

3.1.3 Community Structure of DNS Graph

After building a DNS lookup graph and a DNS failure graph, we want to extract information about the correlations of the vertices and then use this information for detecting fast-flux or domain-flux botnets. We observed that the DNS lookup graph and the DNS failure graph exhibit the community structure mentioned in the work of Y. Jin et al. [23]. Analogous to their findings, both the DNS lookup graph and the DNS failure graph consist of a number of connected components that are isolated from each other. Among these connected components, there are some giant connected components that can be decomposed further into several loosely connected dense subgraphs. Those dense subgraphs represent the intensity in the mapping patterns between domain names and their corresponding IP addresses in the DNS lookup graph, and the intensity in query patterns between domain names and hosts in the DNS failure graph. Figure 3.1 shows an example of the community structure in a DNS failure graph generated using Graphviz [36].

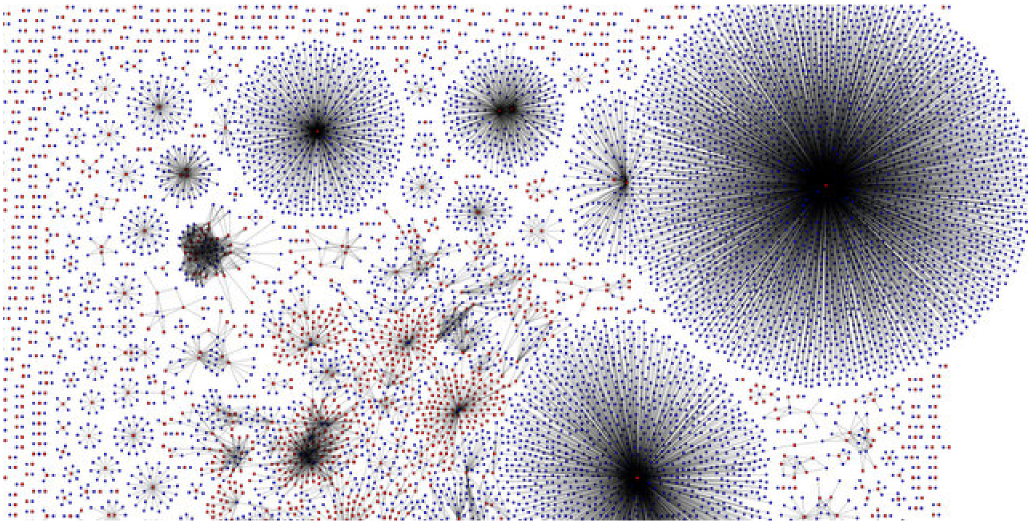


Figure 3.1: An example of community structure in DNS failure graph. A red square represents a host, while a blue circle represents a domain name.

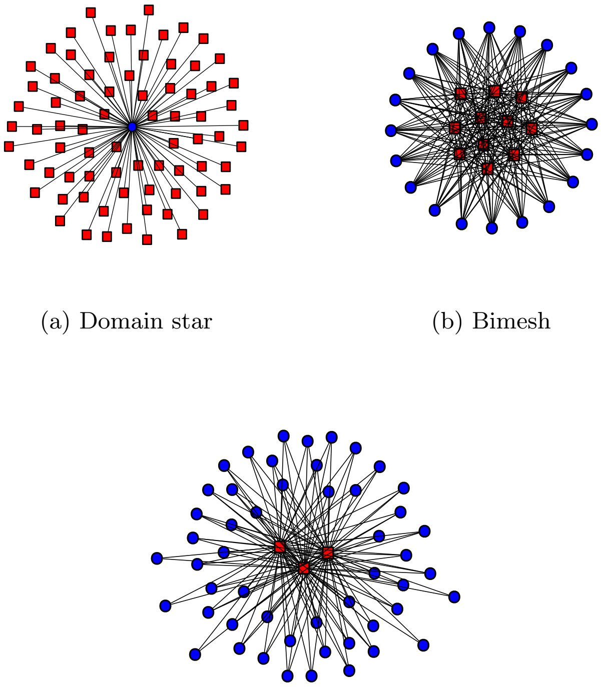
Analogously to the results claimed by Jiang et al. [19] and Jin et al. [23], our investigation shows that the structure of the dense subgraphs fall into three main categories: domain star, IP star, and bi-mesh. As stated in [19, 23] these structures are evidence of the presence of malicious activities in a network.

Domain star: This category contains subgraphs which have some dominant domain names. These domain names are mapped to a large number of IP addresses in the case of a DNS lookup graph, or are unsuccessfully queried for by a large number of hosts in in the case of DNS failure graph.

IP star: In contrast to domain star subgraphs, in IP star subgraphs, there are some dominant IP addresses which are associated with a large number of domain names. In the DNS lookup graph, the same set of IP addresses are returned for multiple domain names, while in the DNS failure graph the same set of host interfaces' IP addresses issued failed DNS queries to a large set of domain names.

Bi-mesh: The subgraphs in this category have approximately an equal number of domain names and IP addresses in the DNS lookup graph (or hosts in the DNS failure graph) which are strongly connected to each other. This type of subgraph shows intensive correlation between domain names and IP addresses in the DNS lookup graph (or hosts in the DNS failure graph).

Figure 3.2 shows examples of different structures of subgraphs extracted from a DNS failure graph.



(a) Domain star

(b) Bimesh

(c) IP star (Host star)

Figure 3.2: Example of structures of subgraphs in a DNS failure graph. A red square represents a host, while a blue circle represents a domain name.

The observation of community structure in the DNS lookup graph and the DNS failure graph leverages the idea of extracting dense subgraphs from these graphs and using the subgraphs's features to discover abnormal behaviors that can indicate malicious activities within a network. As noted in Chapter 2, in a fast-flux botnet, the IP addresses in the DNS responses are changed frequently. Thus, we expected that the dense subgraphs in the corresponding DNS lookup graph will exhibit this correlation between domain names and their associated IP addresses. Conversely, we expected that the correlation between domain names and hosts caused by a domain-flux botnet will result in dense subgraphs in the corresponding DNS failure graph. Using different features extracted from these dense subgraphs as input for a machine learning algorithm allows us to evaluate the probability that these dense subgraphs are fast-flux and/or domain-flux botnets in a network. In the following section, the proposed analysis procedure will be described in detail.

3.2 Analysis Procedure

The analysis procedure for the proposed technique is depicted in Figure 3.3. The overall procedure takes captured network traffic as input and produces a maliciousness score associated with a group of domain names and IP addresses (or hosts) which corresponds to a dense subgraph in the network traffic graph. The prototype system was implemented and attached to the General Packet Radio Service (GPRS) Core Network and is fed with network traffic from a mobile data network in order to detect suspicious activities in the mobile data network. However, this analysis procedure could be deployed in any network which supports TCP/IP, hence it can be utilized with to any ISP's or operator's network.

The traffic data is captured by sensors at the gateway of the GPRS Core Network and then processed with multiple modules. Further details on each module are described in subsequent sections. Below brief descriptions of these modules are given.

(a) **Graph Construction Module:** In this module, DNS traffic data is extracted from the captured data, anonymized, and divided into multiple epochs with a duration of one day ($T = 1\ day$). Afterwards, a DNS lookup graph and a DNS failure graph are built from the DNS query responses as described in Section 3.1. In order to reduce the amount of data to be processed, this module also performs some filtering of the data to remove irrelevant data, such as the DNS queries associated with popular sites[†], DNS overloading [19], server errors, and malformed queries. Note that this filtering process is conservative in the sense that it only removes traffic which is *unlikely* to be due to suspicious activities. A further description of this module is presented in Section 3.3.

(b) **Graph Decomposition Module:** The DNS lookup graph and the DNS failure graph constructed in previous module are input to a graph decomposition module. This module first uses the Breath-First Search (BFS) algorithm to find connected components. Afterwards, those connected components are

[†] We used the list of top 2000 domain names in the ranking system of Alexa [37] as the list of popular sites. The intuition is that although a malicious site can attract a large amount of queries, it is not possible for this site to be one of the top sites as ranked by Alexa.

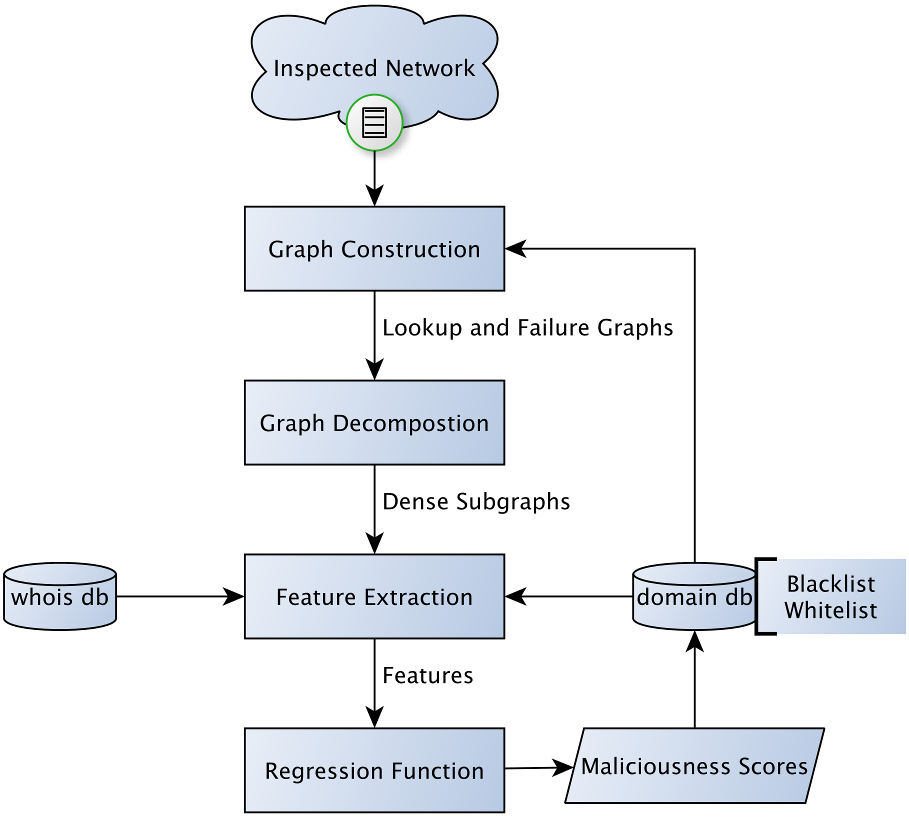


Figure 3.3: Analysis procedure

decomposed into dense, coherent subgraphs using the co-clustering algorithm *Nonnegative Matrix Tri-Factorization* (NMTF) [24]. The resulting subgraphs (or clusters) are passed to the next module for further processing. Details of the clustering process and the NMTF algorithm is given in Section 3.4.

(c) **Feature Extraction Module:** For each cluster from the Clustering Module, a number of network-related features are extracted and computed. These features were selected based on the distinctive characteristics of fast-flux and domain-flux botnets observed in our experimental data. Details of this experimental data and the features that we have utilized are given in Section 4.2 and Section 3.5 respectively.

(d) **Regression Function Module:** The input of this module are clusters and their features as extracted by the previous modules. Utilizing a machine learning approach, this module computes a maliciousness score for each cluster. The greater this score, the more malicious a cluster is considered to be. These scores are stored in a database for future reference. Details of the machine learning algorithm and the ground truth that we used to train this algorithm are given in the Section 3.6.

3.3 Graph Construction Module

This module performs pre-processing of the data and then builds the DNS lookup graph and the DNS failure graph. The detailed steps are described in following subsections.

3.3.1 DNS Data Extraction

The DNS data used for the experiments was extracted from traffic captured using `Tshark` [38]. `Tshark` is a command line tool for network data manipulation and analysis based on the `libpcap` [39] library. `Tshark` provides the ability to capture live network traffic or read a data file saved in `libpcap` format. `Tshark` allows the user to extract the desired data from the traffic using one or more `libpcap` filters. In our experiments, we used `Tshark` to extract all DNS A and NS records in the captured traffic.

The DNS data is then divided into two datasets: successful DNS queries and failed DNS queries. This determination is based on the response code in the DNS query responses. A query with the response code of `NO_ERROR` is considered to be successful, otherwise the query failed [1, 2].

3.3.2 Anonymization

In order to guarantee privacy, we applied an anonymization process to the extracted DNS data before processing the data. This anonymization was performed using *Cryptography-based Prefix-preserving Anonymization* (Crypto-PAn), a tool for anonymizing the IP addresses in traffic traces in a prefix-preserving manner [40, 41]. Crypto-PAn guarantees a one-to-one mapping from the original IP address to an anonymized IP address, while preserves the *prefix* [42] shared between any two original IP addresses, i.e., if two IP addresses in the original trace share a k-bit prefix, their corresponding IP addresses in the anonymized trace will also share a k-bit prefix [40, 41]. Furthermore, Crypto-PAn is a cryptography-based anonymization tool, hence it ensures the secrecy and (pseudo-)randomness of the mapping, while providing a consistent mapping between the original IP address space and the anonymized IP address space across traces *without* requiring a detailed mapping to have been stored, thus the same IP address in the original trace will be mapped to the same anonymized IP address even though it appears in a different trace that is processed at a different time. This property allows Crypto-PAn to operate efficiently. Note that the mapping provided by Crypto-PAn is a one-way mapping, this is a desired privacy property for the anonymization process. However, in our experiment, we used a modified version of Crypto-PAn that allows us to store the mapping between the original IP address to an anonymized IP address in order to be able to reverse the anonymization process. This stored mapping allows a network's operators to determine an IP address in their network that may be infected by a botnet based upon the result of our analysis.

3.3.3 Data Filtering

The data filtering step removes traffic queries for popular sites (e.g. *google.com, facebook.com, amazon.com*), traffic caused by DNS "overloading" or server error, or queries with malformed domain names which is either legitimate or unlikely to be associated with any malicious activities. These types of traffic account for a large amount of DNS traffic in a network and they may introduce noises that affects the accuracy of our analysis process. For this reason, we filtered out this traffic before any further analysis is performed.

The details of the traffic to be filtered are as follows:

(a) **Queries for popular sites:** These are queries for well-known sites operated by well-known services providers such as Google, Facebook, Amazon, or Akamai. In order to remove this traffic, we obtained a list of the top popular sites from Alexa, a well regarded web-site ranking system [37]. Our intuition is that although a malicious site can attract a lot of queries, it is unlikely to be among the top sites listed by Alexa. For this reason, we decided to remove queries from domain names that match subdomains of any domain listed in Alexa's top 2000 sites. A threshold of 2000 is chosen based on observation on the list of popular sites given by Alexa. Furthermore, we skipped all the sites associated with dynamic DNS services because these dynamic DNS services may potentially be used by domain-flux botnets (e.g. Kraken). A Deterministic Finite Automaton (DFA) is built from the top sites listed by Alexa. This DFA allows us to perform fast matching (with computational complexity of $O(k)$; where k is the maximum length of a domain name in Alexa's top sites), rather than linearly checking the list of popular sites each time a domain is checked (which would have a computational complexity of $O(n.k)$; where n is the length of the Alexa's top sites list ($n = 2000$) and k is the maximum length of a domain name in that list).

(b) **DNS "overloading":** *DNS "overloading"* is a new usage of DNS which is employed by anti-spam or anti-virus software to notify the querying host that the requested domain name or IP address is in blacklists that they are maintaining [43, 19]. For example, when a SMTP server receives an email message from an address in the domain `pvrmhn.mooo.com`, it checks if this domain in the blacklist by sending a DNS query to a blacklist service `dnsbl.com` with the form `pvrmhn.mooo.com.dnsbl.com` (or `pvrmhn.mooo.com` depending on the blacklisting service provider). If the domain is in the blacklist, then the DNS server will return an address in the `127.0.0.0/8` network, otherwise it will return a response with a response code of `NXDOMAIN` (meaning Non-Existent Domain) [1, 2]. This mechanism is the same when an IP address is checked, except that the order of the IP address is reversed before concatenated with another fixed domain name (such as “bl.spamcop.net”) to form the domain name in the query to the blacklist service. A detailed example of DNS “overloading” is depicted in Listing 3.1.

Listing 3.1: An example of DNS “overloading”

```
$ dig pvrmhn.mooo.com
...
pvrmhn.mooo.com.        10800   IN      A       127.0.0.2

$ dig 83.22.41.77.bl.spamcop.net
...
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 44794
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
```

In order to filter out DNS “overloading” traffic, we compared the names of authoritative servers and the queried domain names against a list of blacklist services provided by well-known anti-spam and anti-virus organizations such as Spamhaus [44], BitDefender, F-Secure, or McAfee. If the query was found to have the format of a DNS “overloading” query (i.e., ending with any of the domain names in the list of the blacklisting services’ domain names), it was removed from the experimental data. Similar filtering out queries to popular sites, a DFA was also employed to speed up the process of filtering DNS “overloading” queries.

(c) **Server error:** A server error occurs when the domain is temporarily unresolvable because of the failure of the DNS server itself. The responses in this case have a response code of SERVERFAILURE [1, 2], and they usually do not have a response field. In order to filter out this type of traffic, we removed the responses with the response code of SERVERFAILURE and/or those having no data in the response fields.

(d) **Queries with malformed domain names:** A query is considered as a query with a malformed domain name (or *malformed query*) if the queried domain name is an invalid domain name. For example, an IP address or an email address may be used as the domain name in the query, or the domain name is in an invalid format, or does not have a correct TLDs for use in the Internet services listed by the Internet Assigned Numbers Authority (IANA) [45]. To detect such a malformed query, we used a regular expression to check for the format of a valid domain name as specified in RFCs 1034 and 1035 [1, 2]. We also required that the valid domain names must end with one of the TLDs in the list published by IANA [45]. We employed the same approach of using a DFA as in the previous cases to quickly match the TLDs. Any query with a queried domain name that did not meet the requirements to be a valid domain name was removed from the captured traffic.

As we can see, the above data filtering processes are conservative in that they retain all the data related to DNS queries likely to be associated with malicious activities in the network while removing irrelevant or legitimate DNS traffic that might introduce noise into our analysis. Table 3.1 summarizes the data filtering processes applied to our two data sets of successful DNS queries and failed DNS queries.

Table 3.1: Data filtering processes applied to each data set

Filters	Successful queries	Failed queries
Queries for popular sites	Yes	Yes
DNS "overloading"	Yes	Yes
Server error	No	Yes
Queries with malformed domain names	Yes	Yes

3.3.4 Graph Construction

After pre-processing (i.e., DNS data extraction and anonymization) and filtering, the graph construction is straight forward. A DNS lookup graph and a DNS failure graph are built from the data of successful queries and failed queries (respectively). Note that during the graph construction process, the associated attributes of the domain names and IP addresses (or hosts) are also extracted. The program for constructing a DNS lookup graph and a DNS failure graph were written in Python [46]. The program takes the extracted DNS data stored in a textual file format as input, then computes the lookup and failure graphs for each *epoch*. These graphs are output to a file for further processing. These graphs can also be visualized using `pygraph`, a Python open source interfaces to Graphviz [36].

3.4 Graph Decomposition Module

The DNS lookup graph and DNS failure graph obtained from the Graph Construction Module are further processed by the Graph Decomposition Module. This module first performs a Breath-First Search (BFS) on the graph to find the connected components of the graph. For components with a significant number of vertices[‡], the *Nonnegative Matrix Tri-Factorization (NMTF)* [24] algorithm is used to decompose these components into dense, coherent subgraphs. The NMTF algorithm has been claimed to be effective in finding the dense subgraphs of a network traffic graph in previous studies [23, 19]. The overall procedure for graph decomposition is given as Algorithm 1.

Further details of the NMTF algorithm, how this algorithm is used to obtain coherent co-clusters in the graph, and some practical issues with the NMTF algorithm are described in following subsections.

[‡] We considered components with of at least 30 vertices as significant, see Chapter 4 for further details.

```
input  : DNS lookup or failure graph $\mathcal{G}$
output: Dense, coherent subgraphs
Find connected components $\mathcal{G} = \bigcup_i \mathcal{G}_i$ ;
foreach $\mathcal{G}_i$ do
    $\{\mathcal{C}_i\}$ = NMTF($\mathcal{G}_i$);
    foreach $\mathcal{C}_i$ do
        if density($\mathcal{C}_i$) $\geq$ threshold then Output $\mathcal{C}_i$ ;
    end
end
```

Algorithm 1: Graph decomposition procedure

3.4.1 Graph Decomposition

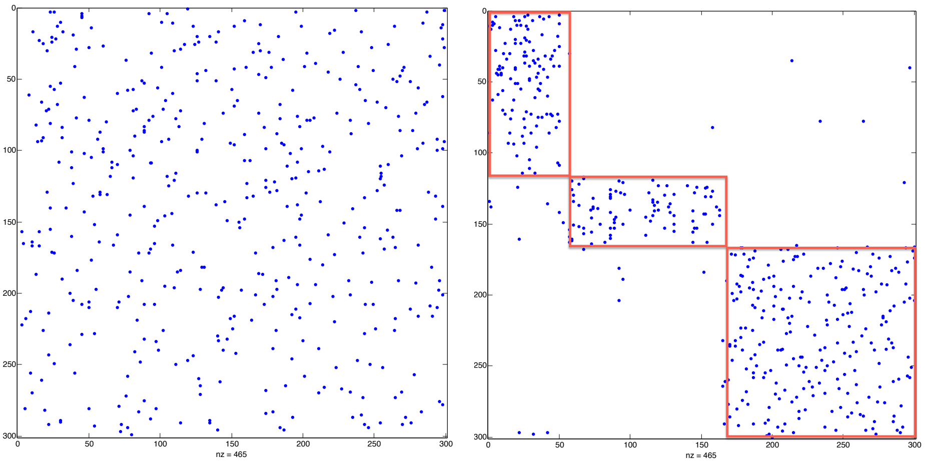
As discussed in the previous sections, the DNS lookup graph and the DNS failure graph both have a community structure. This community structure reveals the communication patterns between domain names and IP addresses (or hosts) in a network. Each community in a graph represents a group of domain names and IP addresses (or hosts) that communicate extensively to each other. Thus, it is desirable to extract those communities out of the graph and perform further analysis on them.

Observations on our data set shows that both the DNS lookup graph and the DNS failure graph are sparse bipartite graphs with an adjacency matrix in the form:

$$M = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$$

where A is a sparse matrix presenting the interaction between domain names and IP addresses (or hosts)[§]. The rows and columns of this matrix can be reordered to reveal some dense blocks in the matrix as illustrated in Figure 3.4. These blocks correspond to the dense subgraphs in a graph. In other words, the problem of decomposing a graph into dense subgraphs is equivalent to extracting these dense blocks in the sub-matrix A. In the next subsection, the NMTF co-clustering method that we used for graph decomposition is described in detail.

[§] For simplicity, from this point on, we use the terms *IP address* and *host* interchangeably when there is no difference between a DNS lookup graph and a DNS failure graph in the discussed topic. Furthermore, in these cases, we also use the term *graph* to specify both a DNS lookup graph and a DNS failure graph.



(a) B

(b) B after reordering

Figure 3.4: Examples of dense blocks in a sparse matrix

3.4.2 The NMTF Co-clustering Method

Analogously to [23], the problem of extracting the dense subgraphs from a DNS lookup graph or a DNS failure graph can be formulated as a *co-clustering* (also known as the *simultaneous clustering*) problem in which the domain names and IP addresses are *jointly* clustered into k domain name groups and l IP address groups. *Co-clustering* has been proved to effectively extract the relationship between the data and its features (i.e., the rows and the columns when the data is organized as a matrix) [47]. These domain name groups and IP address groups form $k \times l$ submatrices, each of which is determined by a pair consisting of a domain name group and a IP address group. *Dense subgraphs* of the original graph can be extracted from the subgraphs represented by these submatrices.

As illustrated in [24], the *co-clustering* problem can be reformulated as a *orthogonal nonnegative matrix tri-factorization* problem, which can be solved using the Nonnegative Matrix Tri-Factorization (NMTF) algorithm. Basically, the *orthogonal nonnegative matrix tri-factorization* is defined as follows: Given a nonnegative matrix $A \in \mathbb{R}_{+}^{m \times n}$, the aim is to factorize (or more precisely, *approximately decompose*) this matrix into three *low-rank nonnegative matrices* $U \in \mathbb{R}_{+}^{m \times k}, S \in \mathbb{R}_{+}^{k \times l}, V \in \mathbb{R}_{+}^{n \times l}$ (i.e. $k, l \ll min(m, n)$) in which U and V are orthogonal matrices.

$$A \approx USV^T \tag{3.1}$$

More precisely, we solve

$$\min_{U \geq 0, S \geq 0, V \geq 0} \|A - USV^T\|^2, \quad s.t. \quad UU^T = I, V^T V = I \tag{3.2}$$

where $\|\cdot\|^2$ is Frobenius norm-2. The *orthogonal* constraints for U and V distinguish NMTF from other algorithms in the *Nonnegative Matrix Factorization* (NMF) family, and enable it to simultaneously cluster the rows and columns of a matrix [47]. As was illustrated in [24] Equation 3.2 corresponds to the equation to find the solution for *simultaneous* K-means clustering of rows and columns of matrix A.

The NMTF solution is obtained using an iterative optimization procedure. The matrix U, S, and V are first initialized to random positive matrices, then the following iterative updating rules are applied to decrease the mean square errors:

$$V_{jk} \leftarrow V_{jk} \frac{(A^T U S)_{jk}}{(V V^T A^T F S)_{jk}} \tag{3.3}$$

$$U_{ik} \leftarrow U_{ik} \frac{(A V S^T)_{ik}}{(F F^T A^T V S^T)_{ik}} \tag{3.4}$$

$$S_{ij} \leftarrow S_{ij} \frac{(U^T A V)_{ij}}{(U^T U S V^T V)_{ij}} \tag{3.5}$$

It has been proved in [24] that with the above updating rules, the algorithm will converge to a local minimum.

3.4.3 Interpretation of NMTF Results

The NMTF result is interpreted in [48] as a probabilistic model that is associated with optimal co-clustering in a sense that loss of mutual information or correlation between row-clusters and column-clusters is minimized. The factor matrices U and V are cluster indicators in which each row contains the probabilities that a row or a column belongs to a cluster. The matrix S shows how the row-clusters are related to column-clusters and vice versa. Figure 3.5 illustrates an example where a matrix $A \in \mathbb{R}_+^{6\times 8}$ is clustered into three row-clusters and two column-clusters. In this figure a larger square indicates the larger value of the corresponding element in the matrix. As one can see the the figure, U and V shown three row-clusters and two column-clusters respectively. Each column in S tells the relationship between row-clusters and column-clusters. For example, in this case, row-clusters 1 and 3 contribute to column-cluster 1, while row-clusters 2 and 3 contribute to column-cluster 2.

In the context of our problem of decomposing the DNS lookup graph and the DNS failure graph, the result is interpreted similar to the approach in [23, 19]. The orthogonal matrices U and V divide domain names and IP addresses into k domain name clusters and l IP address clusters respectively. A pair of a domain cluster and an IP cluster forms a subgraph (or a co-cluster[¶]) in our graph. As

[¶] We will use the term *subgraph* and *co-cluster* interchangeably from this point forward.

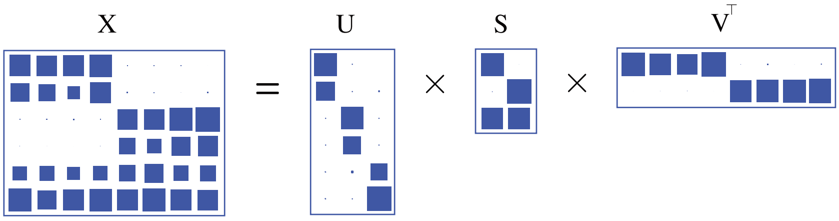


Figure 3.5: An visual interpretation of NMTF result [48]

mentioned above, each element U_{ip} in U indicates the probability that a domain name i belongs to a domain name cluster p. Similarly, each element v_{jq} in V indicates the probability that an IP address belongs to a IP address cluster q. In other words, matrices U and V naturally form a *soft co-clustering* in which a domain name or an IP address can belong to multiple clusters. For simplicity, in the scope of our work, we transformed this soft co-clustering into a *hard co-clustering* in which a domain name or an IP address can belong to only one co-cluster. This is done by assigning a domain name (or an IP address) to only the cluster with the highest probability, i.e., the cluster corresponding to highest value in a row. If all values in a row are zeros, then the domain name or IP address is not assigned to any cluster. Additionally, if there are multiple clusters associated with the highest value, then the domain name or IP address is randomly assigned to one of the candidate clusters. Formally, we transform the matrices U and V into *membership indicator* matrices $\widehat{U}$ and $\widehat{V}$ as:

$$\widehat{U}_{ip} = \begin{cases} 1 & \text{if} \quad p = argmax_j\{U_{ij} : U_{ij} > 0\} \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

$$\widehat{V}_{jq} = \begin{cases} 1 & \text{if} \quad q = argmax_i\{V_{ji} : V_{ji} > 0\} \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

With $\widehat{U}$ and $\widehat{V}$ obtained from above step, we used a generic notation $\mathcal{G} := \{\mathcal{D} \times \mathcal{H}, \mathcal{E}\}$ to denote either a DNS lookup graph or a DNS failure graph. Analogously to [23, 19], we compute the *group density* matrix as:

$$\widehat{H}_{pq} := \frac{(\widehat{U}^T A \widehat{V})_{pq}}{\|\widehat{U}_{\cdot p}\|^1 \cdot \|\widehat{V}_{\cdot q}\|^1} \quad 1 \leq p \leq k, 1 \leq q \leq l. \tag{3.8}$$

where $\|\cdot\|^1$ is Frobenius norm-1. We defined the *density* of a bipartite graph $\mathcal{G} := \{\mathcal{D} \times \mathcal{H}, \mathcal{E}\}$ as:

$$density = \frac{\|\mathcal{E}\|}{\|\mathcal{D}\| \cdot \|\mathcal{H}\|} \tag{3.9}$$

where $\|\mathcal{X}\|$ is the population of the set $\mathcal{X}$. The higher this value is, the more dense the corresponding bipartite graph is. When the density value reaches 1, the bipartite graph become a bipartite clique. Based on the density's definition above, it is apparently that a element $\widehat{H}_{pq}$ represents the density of a subgraph $\mathcal{G}_{pq}$ constructed by the domain name cluster p and host cluster q. A larger value of $\widehat{H}_{pq}$ indicates a strong connection and correlation between the domain names and hosts in the corresponding subgraph.

3.4.4 Obtaining Coherent Co-clusters

Having the group density matrix $\widehat{H}$ as defined in subsection 3.4.3, it is straightforward to obtain dense subgraphs by setting a threshold δ on the density of the subgraphs. A subgraph corresponding to $\widehat{H}_{pq}$ is extracted if and only if $\widehat{H}_{pq} \geq \delta$.

However, in practice, it is difficult to determine the number of domain name clusters k and host clusters l in advance. Multiple methods can be applied to determine appropriate values of k and l such as trial-and-error or statistical testing. These methods require running the NMTF algorithm several times. Unfortunately, the NMTF algorithm has quite high computational complexity. This results in an expensive computation in practice that is undesirable or even impractical when the size of the graph increases. Thus, in our work, we employed a method similar to [19]. Instead of trying to find the "true" values of k and l, we ran the NMTF algorithm with relatively high values[‖] of k and l, which produces a *coarse-grain* clustering result. The co-clusters with high density, i.e., greater than δ, are extracted. Then we merge co-clusters which share either a common domain name cluster or a common host group. This way we obtain the coherent subgraphs represented by irregularly shaped adjacency matrices, rather than just rectangular ones. Formally, we use DG_p to denote pth domain name cluster, HG_q to denote qth host cluster, and $\mathcal{H} := \{\mathcal{DG} \times \mathcal{HG}, \widehat{H}\}$ to denote the (hyper)graph where an edge (DG_p, HG_p) exists between a domain name cluster DG_p and a host cluster HG_p *if and only if* $\widehat{H}_{pq} \geq \delta$. In our experiment, δ was set to, see Chapter 4 for further details. The co-clusters merging process can be formulated as a process of finding the connected components in (hyper)graph $\mathcal{H}$. This process is depicted in the Figure 3.6.

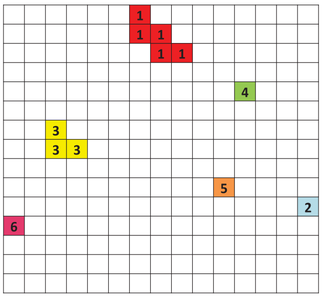


Figure 3.6: Merging co-cluster [19]

[‖] We set $k = l = min(m, n)/30$ in our experiments. Note that this value is based on our observations of the data as described in Chapter 4.

Analogously to the result in [19], this method allows us to obtain dense and coherent subgraphs from both the DNS lookup graph and the DNS failure graph.

3.4.5 Practical Issues

In our proposed method, the NMTF is implemented using the iterative updating process introduced in [24]. The computation time of the algorithm depends on the number of iterations that in turn depends on how the matrices U, S, and V are initialized. In our implementation, we employed the SVD initialization method that is proposed in [49]. The initialization process of the algorithm is as follows: at first, we perform a rank-k singular vector decomposition (SVD) on A, then we projected the rows into a k-dimensional subspace spanned by the top k principal components. Then we performed a k-means clustering on the rows to obtain a row cluster membership indicator matrix as the initial value for matrix U. We set the value of matrix $U = U + \epsilon$ where ϵ is a small positive value in order to avoid zero entries in U. The same process is used to initialize matrix V. The matrix S is initialized as $S = U^T AV$. It has been proved in [49] that this SVD initialization process helps the NMTF algorithm converge faster to the optimal solution.

3.5 Feature Extraction Module

Given the subgraphs from the Graph Decomposition Module, a number of features are extracted for each subgraph. These features are expected to capture the distinctive characteristics of malicious activities in the DNS traffic. Some of the features are only applied to the DNS lookup graph or the DNS failure graph, while others are applied to both types of graphs. The detailed descriptions of all the extracted features are:

(1) Median[**] life-time of a domain name in the subgraphs: The life-time of a domain name is defined as the interval from when the domain name first appears in DNS traffic until the last time the domain name is seen. This feature aims too detect *short-lived* domain names which are often algorithmically generated domain names [26].

(2) Median life-time of an IP address: The life-time of an IP address is analogous to the life-time of a domain name but is based upon an IP address rather than a domain name. Similar to the case of a domain name's life-time, this feature aims to detect *short-lived* IP addresses that are often caused by the rapid swapping of IP addresses in fast-flux botnets [26].

[**] We opted to use median measures instead of mean measures when computing the features for the subgraph because the median measure is less sensitive to outliers than the mean measure is. Furthermore, the size of subgraphs in terms of number of domain names and hosts are relatively small, thus the median value appears to be more suitable.

(3) Domain name/IP ratio: The domain name to IP address ratio is the total number of domain names divided by the total number of IP addresses in the subgraph.

(4) Median number of distinct domain names mapped to an IP address. This feature is proposed based on the observation that in domain-flux botnets, the number of queried domain names relative to the number of IP addresses is relatively large.

(5) Median number of distinct IP addresses mapped to a domain. This feature is based on the assumption that the total number of distinct IP addresses ever mapped to a domain name is relative large in fast-flux botnets.

(6) Median number of IP address returned per query. The botmaster does not have physical control of the bots, thus in order to increase the availability of his/her botnet, the number of IP addresses returned per query is relatively high.

(7) Maximum Time To Live (TTL) value of all domains in the subgraph. Similar to legitimate services such as CDN, in order to be able to rapidly swap IP addresses in and out, the TTL values in the responses are set to relatively low values. However, our experimental traffic was captured behind the recursive DNS (RDNS) server, hence the TTL values were decreased by RDNS server. Therefore, we use the maximum value of the TTL as an estimate of the true TTL value returned by the authoritative DNS server. Another possible solution is to take the average of top N values of the TTL values. However, this value deviates further from the true value of the TTL value. It is also a open problem to determine which value of N to use. Thus we opted to use the maximum value of the TTL values in our analysis.

(8) Network diversity of returned IP addresses: The botmaster does not have the ability to choose the geographical location of the bots. Thus, when the IP addresses of these bots are returned in the A or NS records, they spread over multiple networks and geographical locations. This is different from the IP addresses returned by legitimate services. These IP addresses are usually carefully chosen to be "near" to the user and are often located in a particular IP range of a data center or a hosting service provider. To determine this diversity in the set of returned IP addresses, we used the `whois` look up information provided by the *IP to ASN Mapping* project that is maintained by team Cymru [50]. This feature includes following values:

    - Median number of Autonomous Systems (ASes) in which IP addresses reside.
    - Median number of IP addresses per AS.
    - Median number of countries in which the IP addresses reside.
    - Median number of owners of the IP addresses.

(9) Dominant domain ratio (*ddr*) indicates if there are some dominant domain names in the cluster which connect to a large number of IP addresses. A *ddr* close to 0 means that there are a few dominant domain names that connect

to more hosts than other domain names; while a *ddr* close to 1 means that all the domain names in the subgraph are queried by approximately equal numbers of hosts. This value is computed as:

$$ddr := -(\sum_i p_{i\cdot} logp_{i\cdot})/logm \text{ where } p_{i\cdot} := \sum_j a_{i\cdot}/\sum_{i,j} a_{i,j} \tag{3.10}$$

where $p_{i\cdot}$ is the marginal probabilities of the rows, m is number of domain names, and $A_{m\times n} = \{a_{i,j}\}$ is the adjacency matrix of a subgraph.

(10) Dominant host ratio (*dhr*) indicates if there are some dominant hosts in the subgraph. This value is interpreted analogously to *ddr*.

$$dhr := -(\sum_j p_{\cdot j} logp_{\cdot j})/logn \text{ where } p_{\cdot j} := \sum_i a_{\cdot j}/\sum_{i,j} a_{i,j} \tag{3.11}$$

where $p_{i\cdot}$ is the marginal probabilities of the columns, n is number of hosts, and $A_{m\times n} = \{a_{i,j}\}$ is the adjacency matrix of a subgraph.

The *ddr* and *dhr* together determine the shape a subgraph. This shape may be either an IP star, a domain star, or a bimesh [19].

(11) Lexical features (metrics) on domain labels: The metrics are proposed based on the observation that algorithmically generated domain names are frequently random, thus these metrics aim to capture the characteristics of domain names that appear to humans to be random. These lexical features are proposed based upon previous research [18, 51] and observations on our data. These features includes:

- Symmetric Kullback-Leibler (K-L) divergence metric on unigrams and bigrams of alphanumeric characters in the domain name's labels. We compute the symmetric K-L metric as proposed in [18]:

$$D_{sym}(PQ) = \frac{1}{2}(D_{KL}(P\|Q) + D_{KL}(P\|Q))$$

  where P, Q is two discretized distributions of unigrams or bigrams and D_{KL} is defined as:

$$D_{KL}(P\|Q)) = \sum_{i=1}^{n} P(i)log\frac{P(i)}{Q(i)}$$

- Jaccard Index (JI) between bigrams [18]: The Jaccard Index indicates the similarity between two sets and is defined as:

$$JI = \frac{A \cap B}{A \cup B}$$

  In our context, the lexical metric based on JI is computed as followings: for each test word, we find the candidate words in the database which have at least 75% of bigrams in common with our test word. Then the metric is the average of JI values between the test word's bigram set and each candidate word's bigram set.

- Edit distance: This metric is computed as the average value of all the Levenshtein edit distances between any two domain names' labels in the subgraph.

(12) Query pattern: In a domain-flux botnet, bots tend to have a pattern of periodic and bursty queries, i.e., all the bots in the network periodically query the same domain name within a short time window (this occurs when the bots need to contact the C&C servers). The interval between two consecutive queries within this time window is relatively short (ranging from a few hundredths of a second to a few seconds). In order to detect this pattern of periodic and bursty queries, we employed a method similar to the method proposed in [52]. The procedure can be briefly described as:

(a) For each host in the subgraph, we first compute the inter-arrival time between two consecutive queries.

(b) Then we used K-means algorithms to cluster these inter-intervals.

(c) Afterwards, we examined these clusters to determine if there is any large cluster with a small coefficient of variation (CV) value. This large cluster indicates a periodic group of queries in the DNS traffic.

(d) Furthermore, if there are multiple small clusters with small CV values, then we merge these clusters together into a new cluster. If the new cluster has a small CV value, the same process as in step 12c is applied.

(13) IP overlap: This feature is proposed based on the observation in [31] that there is an overlap between the IP address set returned for A records and the one returned for NS records in a fast-flux botnet. This is due to the fact that a fast-flux has a limited number of bots with high bandwidth and public availability to use as C&C servers, DNS servers, or proxies to these servers. Thus, the botmaster has to swap these bots to have a rapid rate of of change IP addresses. This causes an overlap in the IP addresses returned for A records and NS records. This feature is computed as the percentage of the overlap in the set of IP addresses returned for A records and NS records for domain names in a subgraph.

(14) IP growth rate: This feature reflects how the set of IP addresses returned for a domain name grows per request. It is computed as the number of distinct IP addresses returned divided by the total number of IP addresses returned. As discussed in [31], the set of IP addresses returned for a domain name in fast-flux botnet will continue to increase over time while this set for legitimate services such as CDN usually reaches a limit relatively early. The reason is that in a legitimate service, the servers are rather static and the DNS mappings to them do not change frequently. When the load-balancing mechanism is performed to select a set of IP address that is "near" the users, this set of IP address tends to be stable for a specific user or group of users. While in a fast-flux botnet, the IP addresses associated with the bots are usually dynamic, and the set of IP address to be returned is often chosen randomly among the IP addresses available bots. Thus, this set of IP address spreads over multiple geographical locations.

Table 3.2 summarizes the list of extracted features applied to each type of graph, i.e., a lookup or failure graph.

Table 3.2: Extracted features and their application to each type of graph

No.	**Feature**	**Lookup Graph**	**Failure Graph**
1.	Median domain's life time	Yes	Yes
2.	Median IP's life time	Yes	No
3.	Domain/IP ratio	Yes	Yes
4.	Median number of distinct domains	Yes	No
5.	Median number of distinct IPs	Yes	No
6.	Median number of IPs per query	Yes	No
7.	Median maximum TTL value	Yes	No
8.	Network diversity of return IPs	Yes	No
9.	Dominant domain ratio	Yes	Yes
10.	Dominant host ratio	Yes	Yes
11.	Lexical features on domain labels	No	Yes
12.	Query pattern	No	Yes
13.	IP overlap	Yes	No
14.	IP growth rate	Yes	No

3.6 Regression Function Module

The Regression Function Module takes dense subgraphs and their extracted features as input. Due to the difference in the ranges of the values of the features, we first standardized our data using `zscore` function in MATLAB®. In another words, we treated all the features with equally importance level. Afterwards, using a machine learning approach we expect to be able to distinguish malicious subgraphs from benign subgraphs by assigning to each subgraph a maliciousness score in the range $[0, 1]$. This maliciousness score indicates the probability that a subgraph is malicious. In reality, a threshold can be set so that the system will raise a warning when a subgraph has a maliciousness score greater than the chosen threshold.

In our prototype system, we decided to use supervised machine learning algorithms in our Regression Function Module. A common task when working with a machine learning approach is to select the best algorithm from a list of candidates based on a goodness metric estimated by a cross-validation process. The best algorithm (or model) is the one that maximizes the model's goodness metric [53].

The candidates for our model selection process were following[††]:

- Artificial Neural Network (ANN)
- Decision Tree (DT)
- Decision Tree with AdaBoost (DTA)
- Linear Regression (LR)
- Linear Regression with AdaBoost (LRA)
- Logistic Regression (LOG)
- Logistic Regression with AdaBoost (LOGA)

We used the F-score [53] as the goodness metric in our model selection procedure. We labeled data using a blacklist obtained from different sources (the details of which will be described in Chapter 4). A *10-fold cross-validation* [53] process was employed to compute the average goodness metric over all of our data. The *Decision Tree with AdaBoost* algorithm gave the best goodness metric, thus it was selected as our regression function in our prototype system. Additionally, we employed a *sequential forward feature selection* [53] procedure to determine significant features and to remove noisy features from our feature set. Additional detail of the result of this model selection and feature selection process will be described in Chapter 4

[††] Note that some of our candidate algorithms are classification algorithms, i.e., they only produce a result of 0 or 1 which corresponds to the two classes *benign* and *malicious* of our data, thus here we loosely consider the term *regression* in order to include these algorithms in our candidate set.

Chapter 4

Results and Discussion

In this chapter, we described the data collected, the experiment scenarios, and the F-score that we used to measure the quality of our detection technique. Finally, the experimental results are presented and discussed.

4.1 Prototype Implementation

We developed a prototype system for the experiments with our analysis technique. In addition to some components of existing open sources tools and libraries, new code was written in Python 2.7.2 and MATLAB® 2010b to complete our prototype. Python and MATLAB® were used because they are suitable programming languages for rapid prototyping. For the machine learning algorithms, we used the MATLAB® code from Mørup and Schmidt's 02450Toolbox [54]. Table 4.1 summarizes the implementation languages and status of the principal components in our prototype system. A status of `Reused` means that a component was reused *without* any modification, while a status of `Modified` means that a component was modified to meet our purposes. Furthermore, a status of `New` indicates that a component was newly developed.

Table 4.1: Summary of the implementation of the prototype system

Module	**Languages**	**Libs+Tools**	**Status**
Extract DNS data	C	`tshark`	Reused
Anonymization	C	TraceAnon	Modified
Graph Construction	Python	-	New
Data Filtering	Python	DFA	New
Graph Decomposition	Python+MATLAB	mlabwrap	New
Feature Extraction	Python	Cymruwhois	New
Regression Function	Python+MATLAB	mlabwrap+Toolbox	New
Visualization	Python	Graphviz+pygraph	New

4.2 Data Collection

This section describes how we collected the traffic data and how we labeled this data in order to use it as input to our analysis.

4.2.1 Traffic Data

The data collection used in our experiments consists of two parts: captured traffic traces from an operator's network and synthetic bots' traces that we generated and merged into the real traces.

The traces include 25 days worth of traffic data collected from a mobile network, during the period from September 9, 2011 to October 3, 2011. These traces were captured in an operator's GPRS Core Network. The sensors were placed behind the recursive DNS servers in the GPRS Core Network. These sensors captured a copy of all traffic that goes through these recursive DNS servers. In our analysis, we only consider the DNS information, thus we decided to decapsulate the GPRS Tunneling Protocol (GTP) data and then strip off the GTP header from the captured traffic. This is done by a program written in C based on the `libpcap` library [39].

In order to have more data for our experiment, we generated additional synthetic traces of a fast-flux botnet and a domain-flux botnet (which mimics the DGA of the Conficker bot [55, 56]). These synthetic traces are time-shifted and merged into the collected traces. The DNS data were extracted from the merged traces, anonymized, and then split into two datasets of successful queries and failed queries. These two datasets were used to build the DNS graphs as was described in Section 3.3.

4.2.2 Data Labeling and Filtering

In order to label and filter our datasets, we employed both automated and manual methods of data labeling and filtering. For automated labeling and filtering, we collected whitelist and blacklist entries from several sources and matched the domain names in our data against these lists.

Whitelist: To construct the whitelist, we selected the 2000 most popular sites listed in Alexa's ranking system [37]. Note that we removed all the dynamic DNS and URL shortener provider sites from the whitelist because these domain names are likely to be abused by botnets. This white list is used to filter out legitimate or unlikely malicious traffic from the traces in order to reduce the noise of the data that is input to our analysis. Additionally, we collected a list of well-known anti-virus and anti-spam service providers in order to filter out the

DNS "overloading" traffic as described in Section 3.3. This whitelist was also used to build the unigram and bigram database of "good" domain names' labels that is used to calculated the lexical features of the domain names' labels as described in subsection 3.5 and by Yadav, et al. [18].

Blacklist: The blacklist was constructed from multiple sources. These sources were:

- A list of malicious domain names from serveral blacklist services, such as MalwareDomainList [57] and DNS Blackhole [16].
- A list of domain names generated by well-known botnets such as Zeus [58], Conficker [59], Kraken [17], and the synthetic bots that we generated.
- We also extracted the domain names in the body of the spam and phishing emails* collected from Spam Archive [60] and PhishTank [61].

The domain names in the subgraphs are checked against this blacklist. A match occurs when a domain name is exactly matched or is a subdomain of a domain name in the blacklist. A maliciousness score of a subgraph is then calculated as the portion of its domain names in the blacklist, i.e.:

$$mal\_score = \frac{\text{Number of matches}}{\text{Total number of domain names}} \tag{4.1}$$

The blacklist is also used to compute the unigram and bigram of malicious domain names' labels that will after be used to calculated the K-L and JI of domain names' labels of a subgraph – described in subsection 3.5 and in [18].

Manual checking was also performed. We checked the domain names of the subgraphs and looked for those that have strange and random looking domain names. Then we checked these domains against multiple sources including Norton Safe Web [62], SiteAdvisor [63], or Web of Trust [64] to figure out if the domain names are malicious.

Note that our labeling process is a thorough investigation process, a subgraph is not labeled as malicious based solely on the randomness of its domain names' labels. Beside the random-looking domain names, many other factors were considered such as query patterns of the hosts within a subgraph, reputation of the registrars where domain names were located, and reputation of the domain names themselves. We need to check carefully and obtain good reasons before flagging any subgraph as malicious. Thus, the labeling process is one of the most time-consuming tasks in this thesis project.

*These domain names are related to fast-flux because they are often hosted on a FFSN infrastructure created by a (part of a) botnet. Thus, we included these domain names in the blacklist although they may not resolve to a botnet's C&C server(s).

4.3 Experimental Scenarios

Our 25 days worth of data was divided into two datasets in our experiments:

- **A training dataset** corresponding to 20 days data (the first 80% of the total data).
- **A testing dataset** corresponding to 5 days data (the last 20% of the total data).

The experiments are performed in two scenario: offline and online. These two scenarios are described in the next two subsections.

4.3.1 Offline Scenario

In this scenario we labeled the data in the training dataset, then we run a 10-fold cross-validation on this dataset to estimate the goodness metrics of following machine learning algorithms [53]:

- Artificial Neural Network (ANN)
- Decision Tree (DT)
- Decision Tree with AdaBoost (DTA)
- Linear Regression (LR)
- Linear Regression with AdaBoost (LRA)
- Logistic Regression (LOG)
- Logistic Regression with AdaBoost (LOGA)

This scenario allowed us to select the learning algorithm that produced the highest goodness metric to use in the subsequent online scenario. In our experiment, we used the F-score as the goodness metric. The F-score is described in detail in the next section.

A *sequential forward feature selection* [53] procedure was also performed in this step in order to remove any noisy features and to reduce the dimension of the data.

4.3.2 Online Scenario

After obtaining the best model from the offline scenario, we ran an experiment with this model using the testing set consisting of 5 days worth of traffic that was not yet labeled. The purpose of this experimental scenario was to evaluate how our technique perform in practice.

4.4 Goodness Metrics

In order to evaluate the performance of our proposed technique, we need to introduce a goodness metric for a quantitative measurement. Although the goal of our technique is to produce a maliciousness score for subgraphs, in our evaluation we considered that our data are classified into two classes: *benign* and *malicious*. This classification scheme can be easily obtained by setting a threshold δ on the maliciousness scores of the subgraphs. This technique of selecting a threshold is reasonable in a sense that in practice, we also need to set a threshold at which our system should raise a warning about a malicious subgraph. Thus, the evaluation of our technique is similar to the evaluation of a binary classification with *malicious* and *benign* classes corresponding to *positive* and *negative* cases respectively. In our data, the number of malicious subgraphs are smaller than the number of benign subgraphs. Moreover, the correct detection of malicious class is more important that of benign class. Thus, we employed the *F-score*, because it is suitable for the case when there is imbalance between two classes [53]. The F-score definition is derived from the definitions of *Precision* and *Recall*. These definitions are

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

F-score

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FN + FP}$$

where True Positive(TP), False Positive (FP), and False Negative (FN) are defined as:

Precision represents the proportion of subgraphs that are actually malicious in all subgraphs that are predicted as malicious. The higher the precision is, the lower the number of false positive errors produced by the classifier. *Recall* reflects the proportion of subgraphs that are predicted to be malicious from all the actually malicious ones. Recall is equivalent to the true positive rate of the classifier.

Table 4.2: Definition of TP, FP, and FN

	Actual class	**Predicted class**
True Positive (TP)	Malicious	Malicious
False Positive (FP)	Benign	Malicious
False Negative (FN)	Malicious	Benign

There is often a inverse relationship between precision and recall when building a classifier, where it is possible to increase one at the cost of reducing the other. A key challenge in machine learning is to build a model that maximize both precision and recall. Thus, the *F-score* was introduced to present a weighted harmonic mean of both precision and recall [53]. A F-score value close to 1 means that both precision and recall are relatively high, hence the technique produces a good detection result.

4.5 Results

Both training and testing datasets were processed through the procedure described in Section 3.2. In total, 50 graphs are built, a DNS lookup graph and a DNS failure graph were computed for each day of this dataset. Table 4.3 summarizes the 95% confidence interval of the number of domain names, hosts, and edges in a lookup graph and a failure graph for one day of data.

Table 4.3: Summary statistics of graphs (with a confidence interval of 95%)

Graph	**Domains**	**Hosts**	**Edges**	**Density**
Lookup	97870±4686	56528±2076	150974±6904	0.000028±0.000001
Failure	2692±271	1407±79	4002±316	0.001087±0.000064

Using a breath first search to find the connected components of the graphs, we found that although there are some large components with a significant number of vertices, a major portion of the components were small ones that contain only a few domain names and a few hosts. In the DNS lookup graph these components are related to some small sites that are not very popular and that are mapped to a stable set of only a few IP addresses. While in the DNS failure graph, these small components are DNS failures caused by issues associated some particular sites being down or are due to misconfigurations. Thus, we do not look into those small components, but rather we opted to set a threshold min_comp on the number of vertices in order to filter out these small components from our further analysis. We set $min\_comp = 10$ in our experiments. Note that this number is based on our observations on the data and our interest in accurately detecting fast-flux and domain-flux.

Table 4.4 shows the total number of connected components and the number of remaining components after removing all small connection components from our data with the confidence interval of 95%.

Table 4.4: Remaining connected components of graphs after removing the small components (with $min\_comp = 10$ and a confidence interval of 95%)

	Components	
Graph	Remaining/Total	Percentage (%)
Lookup	1257±64/46449±1911	2.70±0.03
Failure	33±3/702±32	4.73±0.40

Table 4.5 shows the remaining percentages of domain names, hosts, and edges after removing small connected components. Even though the remaining percentages of the number of connected components are small, the remaining percentages of domain names, hosts, and edges are high. Thus, the remaining components have relatively large size (in terms of the number of domain names, hosts, and edges) compared to the ones that have been removed.

Table 4.5: Percentages of the remaining domain names, hosts, and edges (with a confidence interval of 95%)

	Percentage (%)		
Graph	Domains	Hosts	Edges
Lookup	35.76±0.54	13.26±0.34	54.97±0.40
Failure	61.55±2.24	41.88±1.38	70.37±1.53

After removing the small connected components, among the remaining components, there are some with a large number of vertices as shown in Table 4.6. This result is analogous to the community structure discussed in Section 3.1.

Inspecting the connected components, we found that most of the components with relative small numbers of vertices are already dense and coherent bipartite graph themselves. Thus, we opted to apply the graph decomposition only to components that have a significant number of vertices. In our experiments, we set this threshold to 30 ($min\_decompose = 30$) based on our observations of the data and our expectation of having subgraphs with a reasonable number of vertices due to our interest in detecting fast-flux and domain-flux. For the same reason, we set $k = l = min(m, n)/30$ as the coarse-grain number of subgraphs in our NMTF algorithm. The graph decomposition's result shows that with this setting of coarse-grain number of subgraphs and merging process, our Graph Decomposition Module successfully extracted the dense and coherent subgraphs

Table 4.6: Ten largest connected components of a failure graph ($min\_comp = 10$)

Comp#	# Domains	# Hosts	# Edges	Density
1	845	536	2280	0.0050
2	7	92	98	0.1522
3	66	3	68	0.3434
4	62	2	91	0.7339
5	33	5	72	0.4364
6	35	1	35	1.0000
7	28	6	56	0.3333
8	27	1	27	1.0000
9	27	1	27	1.0000
10	22	1	22	1.0000

from the giant components. Listing 4.1 shows an example extracted subgraph in a DNS failure graph.

Listing 4.1: An example extracted subgraph from a failure graph

```
sfcg023a.ad1.seb.net.
sfcg023a.corp1.ad1.seb.net.
sfcg023a.seb.net.
smcg093v.ad1.seb.net.
smcg093v.corp1.ad1.seb.net.
smcg093v.seb.net.
smcr038v.ad1.seb.net.
smcr038v.corp1.ad1.seb.net.
smcr038v.seb.net.
smcr090v.ad1.seb.net.
```

Given the extracted subgraphs, we computed their features and ran the experiments in the two scenarios described in Section 4.3.

Note on the lookup graphs: During our experiments, we checked the domain names against the blacklist, but we did not find any matches in our lookup graph. This lack of data makes it difficult to train our machine learning method during training phase, even though we had a bot examples that we generated. Another reason is that although our data was collected from September to October, 2011, when we performed `whois` lookup for IP addresses, some of the `whois` information was already out of date. Furthermore, previous research [6, 25, 26] produced relatively good results by building domain name reputation systems. Thus, at this point in our project, *we opted to only concentrate on running the experiment on the failure graphs, and to focus on the detection of the domain-flux botnets.*

4.5.1 Offline Scenario

Using both automated and manual methods for labeling data as described in Section 4.2, we discovered some domain-flux botnets in our training dataset, and some other suspicious subgraphs. Note that flagging these subgraphs as malicious were based on a thoroughly investigation, not solely based on the fact that the domain names of these subgraphs were random-looking as mentioned in Section 4.2. Table 4.7 summarizes these subgraphs that were labeled as malicious in our training dataset. These subgraph have random-looking domain names with different lengths as shown in the first column by their domain name patterns. For example, the domain name pattern `[a-z]{5-11}.ws` means that these are domain names that have the TLD of `ws`, and the second-level domain labels are random strings containing from 5 to 11 letters. The second column shows a sample of the domain names in the subgraphs, and the third columns shows the number of subgraphs with these patterns.

Table 4.7: Botnet related and suspicious subgraphs in training dataset

Domain pattern	**Sample domains**	#	Note
www.[a-z]{5-8}.info www.[a-z]{5-8}.com www.[a-z]{5-8}.cc www.[a-z]{5-8}.ws www.[a-z]{5-8}.cn www.[a-z]{5-8}.org	www.aucyuod.info www.acromtohk.com www.ahdlmwu.cc www.azlbx.ws www.aplwaci.cn www.vwplcy.org	23	Synthetic
[a-z]{12}.com	ucozktgezixg.com	2	
[a-z0-9]{7}.co.tv [a-z0-9]{7}.uni.cc [a-z0-9]{7}.hopto.org [a-z0-9]{7}.one.pl	ibye6ig.co.tv ntcgoyv.uni.cc titvm.hopto.org vdspnxf.one.pl	1	
[a-z]{5-11}.ws	btrstpcxmw.ws	1	
[a-z]{5-10}.com [a-z]{5-10}.net [a-z]{5-10}.ws [a-z]{5-10}.cc	twxjhaezo.com cypuovqq.net uuikdtzw.ws uxkwcdrz.cc	1	Conficker?
[a-z]{6-11}.dynserv.com [a-z]{6-11}.yi.org [a-z]{6-11}.mooo.com [a-z]{6-11}.dyndns.org	dexklv.dynserv.com dfdblcu.yi.org exotxrdxj.mooo.com dhsmkdf.dyndns.org	1	Kraken

The first row in Table 4.7 are the synthetic bots that we generated and mixed into the captured traffic. The remaining were labeled using different sources. The last row shows the Kraken bot found in the trace. The second last row is a bot that has very similar behaviors to Conficker bot's. The other malicious subgraphs were inspected and verified manually.

As mentioned in Section 3.6, we standardized the values of the extracted features before applying a *Principal Component Analysis* (PCA) process to the training dataset. Figure 4.1 shows the fraction of the variation in the training dataset explained by principal components.

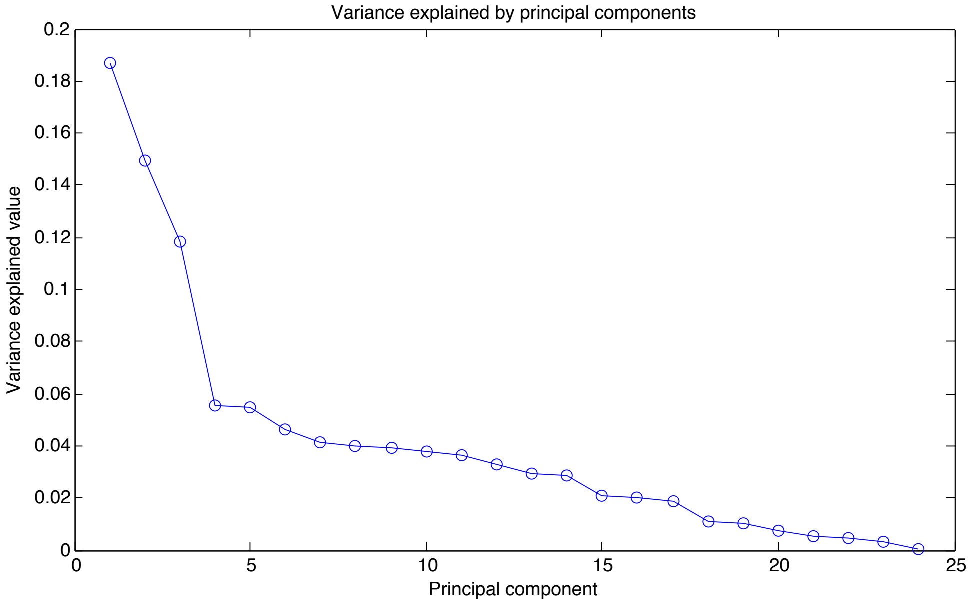


Figure 4.1: Variance explained by principal components of the training dataset

The coefficients of the principal components show that some features such as lexical features on domain name labels, domain names to hosts ratio, and query patterns (see Table 3.2) account for significant portion of the variation in the training dataset. Inspecting the regression function algorithms, we observed that these algorithms could split the training dataset into two classes of benign and malicious subgraphs using some of the projections of the proposed features into the subspace of principal components. Thus, we performed a *sequential forward feature selection* procedure, a standard procedure in machine learning to determine the significant features and noisy features in a feature set. The result showed that we cannot remove any of the features. This means that our selected feature set does not contain any noisy features. Thus, although some features such as lexical features on domain name labels, domain names to hosts ratio, and query patterns account for significant portion of the variation in the data, these features alone were not sufficient to classify our training data. In other words, other features beside these features also contribute in determining whether a subgraph is malicious.

All the labeled training set was used for a 10-fold cross-validation procedure to select the machine learning algorithm and estimate the F-score. The detailed F-score for each model is depicted in Table 4.8.

Table 4.8: F-score for each model

Fold	ANN	DT	DTA	LR	LRA	LOG	LOGA
1	0.6667	1.0000	**1.0000**	0.6667	0.1000	0.6667	0.1000
2	0.6667	0.8571	**1.0000**	0.8571	0.2667	0.8571	0.2667
3	1.0000	1.0000	**0.8000**	1.0000	0.0714	1.0000	0.0714
4	0.7273	1.0000	**1.0000**	0.7500	0.1500	0.7500	0.1500
5	0.6000	0.4444	**0.7500**	0.3333	0.3478	0.3333	0.3478
6	0.8571	1.0000	**1.0000**	1.0000	0.1429	1.0000	0.1429
7	0.6667	1.0000	**1.0000**	1.0000	0.2963	1.0000	0.2963
8	0.6667	0.6667	**0.7500**	0.4000	0.2609	0.4000	0.2727
9	1.0000	0.6667	**1.0000**	0.8000	0.1111	0.8000	0.1081
10	0.8000	0.8000	**0.7273**	0.8000	0.2222	0.8000	0.2222
Mean	0.7651	0.8435	**0.9027**	0.7607	0.1969	0.7607	0.1978

The result of the cross-validation procedure shows that the *Decision Tree with AdaBoost* (DTA) is the model that provides the highest average F-score of `0.9027`. A t-test showed that the F-scores produced by this algorithm (DTA) is significantly different from the F-scores produced by the second algorithm – Decision Tree (DT). Thus, we selected this algorithm for our Regression Function Module. The DTA is an extended version of the original DT algorithm in which there are multiple decision trees instead of only one decision tree. Note that this algorithm computes the weighed average of the scores produced by all the decision trees, therefore the final score is a continuous value in the range of $[0, 1]$. Thus, our requirement of a continuous maliciousness score is satisfied with this algorithm.

4.5.2 Online Scenario

The selected model with Decision Tree with AdaBoost algorithm was used to run the experiment on our test dataset that was not labeled. The result of this experiment shows that our model is able to detect some malicious subgraphs in the test dataset. We configured our experiment so that it raises positive detection results when a subgraph has a *maliciousness score* greater than `0.5`. The subgraphs that were detected by our technique are listed in the Table 4.9.

We can see in Table 4.9 our technique was able to detect all the synthetic bots in our test dataset (the first group of six malicious subgraphs found in Table 4.9). Furthermore, another suspicious subgraph was also flagged as malicious (the second group of one subgraph found in Table 4.9).

An interesting case in the detected subgraph is the `az.pl` and `co.cc` domain

Table 4.9: Subgraphs detected as malicious in the test dataset (with a warning threshold of 0.5)

Domain pattern	**Sample domains**	#	Note
www.[a-z]{5-8}.info www.[a-z]{5-8}.com www.[a-z]{5-8}.cc www.[a-z]{5-8}.ws www.[a-z]{5-8}.cn www.[a-z]{5-8}.org	www.knubgcnvh.info www.kzglnyjo.com www.hzafr.cc www.gazmni.ws www.fxiyasmv.cn www.khuwewv.org	6	Synthetic
[0-9]{1-4}.[a-z0-9]{4-11}.az.pl [0-9]{1-4}.[a-z0-9]{4-11}.co.cc	145.rickenbacker.az.pl 0.rypeygf.co.cc	1	Unknown

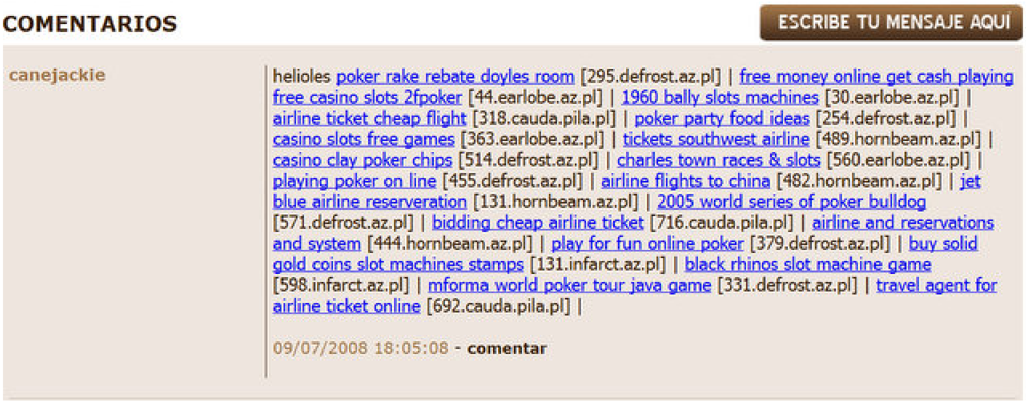
names (shown is the Listing 4.2). All the domain names in this subgraph are subdomains of the second level domain name `az.pl` or `co.cc`. The third labels of these domain names appear to be random and are likely generated by a DGA.

Listing 4.2: Domain names of a detected subgraph with `az.pl` and `co.cc` domain names

```
0.rypeygf.co.cc.         183.ecphonesis.az.pl.        873.pblr1j.az.pl.
1.aiyg.co.cc.            1838.k8mmr.az.pl.            882.xg7fai2.az.pl.
10.pytovkj.co.cc.        1859.6t9she.az.pl.           884.oaflh02.az.pl.
102.9wmo.az.pl.          19.7h9vszb.az.pl.            885.vuer92.az.pl.
18.16st.az.pl.           190.ramiro.az.pl.            890.zzxool5.az.pl.
1804.ecuk.az.pl.         191.contour.az.pl.           891.qxhs1xl.az.pl.
1806.39bz6u9.az.pl.      1941.0545mnu.az.pl.          90.dogori.az.pl.
```

Further investigation led us to a forum on the Internet. The posts on this forum look like a list of instructions for a botnet to conduct fraudulent manipulation of search engine's results in order to redirect users to malicious sites hosted on a botnet. Thus, we have very good reason to classify these domain names as malicious. Figure 4.2 shows an example of a message posted on this forum.

Figure 4.3 depicts the detection result in the subspaces of the first two principal components. We can see from this figure that our synthetic bots appear to stick out further from the benign group compared to other newly detected malicious ones (these are marked with the purple circles). This suggests that we need to refine our generated bot to better mimic the behaviors of a real botnet. Nevertheless, this result confirms that our synthetic bot has succeeded at a certain level in mimicking a real bot. Additionally, our technique produced good detection result.



COMENTARIOS

canejackie

helioles poker rake rebate doyles room [295.defrost.az.pl] | free money online get cash playing free casino slots 2fpoker [44.earlobe.az.pl] | 1960 bally slots machines [30.earlobe.az.pl] | airline ticket cheap flight [318.cauda.pila.pl] | poker party food ideas [254.defrost.az.pl] | casino slots free games [363.earlobe.az.pl] | tickets southwest airline [489.hornbeam.az.pl] | casino clay poker chips [514.defrost.az.pl] | charles town races & slots [560.earlobe.az.pl] | playing poker on line [455.defrost.az.pl] | airline flights to china [482.hornbeam.az.pl] | jet blue airline reserveration [131.hornbeam.az.pl] | 2005 world series of poker bulldog [571.defrost.az.pl] | bidding cheap airline ticket [716.cauda.pila.pl] | airline and reservations and system [444.hornbeam.az.pl] | play for fun online poker [379.defrost.az.pl] | buy solid gold coins slot machines stamps [131.infarct.az.pl] | black rhinos slot machine game [598.infarct.az.pl] | mforma world poker tour java game [331.defrost.az.pl] | travel agent for airline ticket online [692.cauda.pila.pl] |

09/07/2008 18:05:08 - comentar

Figure 4.2: An example message posted on the forum related to az.pl and co.cc domain names

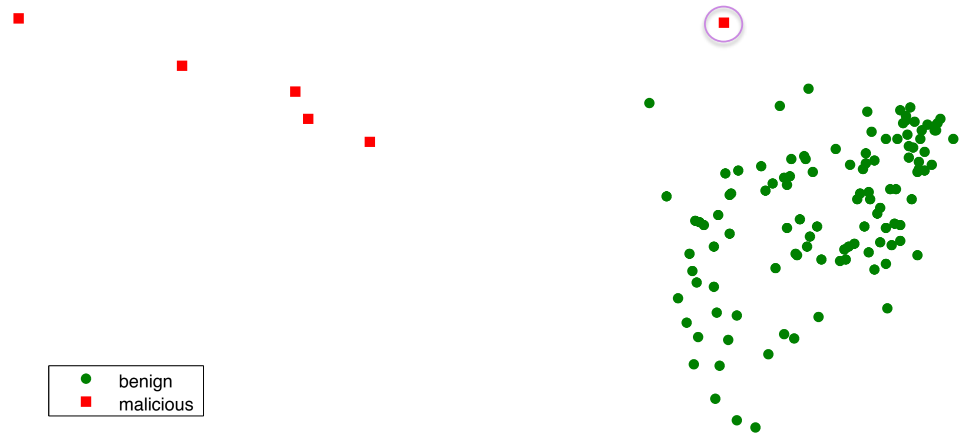


Figure 4.3: Visualization of the detection results in the subspace of the first two principal components. A red square represents a malicious subgraph, while a green circle represents a benign subgraph. Note that one malicious subgraph (circled in the figure) is near the benign subgraphs.

4.6 Discussion

As mentioned in Section 4.5, we opted to put aside the lookup graphs and only focus on the failure graphs for several reasons (in addition to the limited time available for our project). These reasons included:

- Checking against the blacklist did not return any matches. Even though we generated traffic that mimics a fast-flux botnet, this data represents only one botnet, and there are not enough malicious examples to train our regression function and to perform cross-validation.
- Our data was collected from September 9, 2011 to October 3, 2011 and when we perform our analysis in March, 2012 the whois data corresponding to the IP addresses in the traces were out of date. This outdated data had a

strong influence on the analysis because one of the important characteristics of fast-flux botnets is that they have very high diversity in their IP address set. We contacted ISC DNSDB to get access to the Global Passive DNS (pDNS) [65] in order to learn about the mapping in the past of these domain names and IP addresses. However, we did not succeed in getting access to this data.

- Previous research [6, 25, 26] produced impressive results by building domain name reputation systems. These results to some extent have included fast-flux botnet detection. Furthermore, it has been shown in this earlier research that access to a high level of the DNS hierarchy plays an important role in generating meaningful results. Furthermore, information of the domain names from registrars such as the `whois` information of the domain names provides significant evidence. In our case, we did not have access to such high level of the DNS hierarchy. We did not have access to information of the domain names from registrars either. Even though we can perform an active probes to get this information, it is a time-consuming task. Thus, we are focused on the open question of detecting domain-flux botnets from the DNS failure graph.

While working on our project, we found that one of the biggest obstacles when proposing an analysis technique for detecting botnets or malware in general is the lack of malware examples. The lack of observations of how the malware actually behaves makes it difficult to produce a good technique. Even though we generated synthetic bots in our work, this limited number of examples does not represent the diversity in the behaviors of different botnets in the wild. Thus, a malware example collection process should be performed in parallel with development of new detection techniques, either by using a honeynet or artificially generating meaningful and representative data. Furthermore, one of the most time-consuming task in our work is to manually label the dataset for training purposes. Although some subgraphs appear to have similar characteristics to a fast-flux or domain-flux botnet, classifying them as malicious still requires evidence and this requires further (manual) investigation. Thus, it is necessary to develop a reliable and automated technique for labeling the collected dataset.

Despite the good results in detecting domain-flux, the proposed technique also has some limitations. It may produce a false negative when the number of domain names in a malicious subgraph is relatively small. In this case, the malicious characteristics exhibited by the subgraph are not distinctive enough for the technique to capture. For example, it is difficult to identify the randomness of the domain name labels if the number of domain names are few. On the other hand, the proposed technique may produce a false positive when a benign subgraph contains a large number of random-looking domain names.

Chapter 5

Conclusions and Future Work

In this project we have proposed a technique for detecting a domain-flux botnet in a network by analyzing DNS data traffic, particularly the failed DNS queries. This thesis is, according to the author's knowledge, the first thoroughly investigated method that has been proposed to detect the domain-flux based on DNS analysis. Although there was some earlier related research [18, 19], these researchers did not propose any specific procedure to detect malicious activities based upon the failed DNS queries.

The main contributions of this thesis are:

(1) Proposed and thoroughly investigated a new method to detect domain-flux and fast-flux by analyzing DNS traffic.

(2) We have taken into consideration the correlation between domain names and IP addresses by using the concept of a dense subgraph. This allows the proposed method to capture one of the most distinctive features of the fast-flux and domain-flux botnets, that is the strong correlation between domain names and IP addresses.

(3) The thesis provides a comprehensive list of features and a procedure to apply machine learning approaches in order to detect malicious activities related to fast-flux and domain-flux botnets.

(4) An experimental evaluation of the proposed technique has been made using the DNS failure graph. The proposed technique was able to successfully detect domain-flux botnets in experimental traces.

The results of our proposed technique can be easily integrated in any NIDS without significant effort. For example, a set of rules can be derived from the malicious domain names flagged by our techniques. This set of rules can be configured in a NIDS to block these domain names.

Due to some limitations, the analysis was only performed on the DNS failure graph. We decided to leave the experiments on DNS lookup graph for future work.

Within the limited time of the project, there still some open problems that the author has not considered in this thesis. These are possible topics for future research:

- To collect longer traces and develop a reliable technique for automatically labeling in order to have more malicious examples. Alternatively one needs to develop a technique for generating meaningful synthetic botnet data. The goal is to build an experimental dataset that represents the diversity of behaviors of different botnets. This dataset could be used to evaluate the technique proposed in this thesis for using the DNS traffic to detect fast-flux botnets. Furthermore, a technique to reliably and automatically label a dataset is desirable. It is also interesting to mix the data in the training dataset in different order other than the chronological order to see how this mixing would affect the detection model.

- To study further and propose a technique to make NMTF more robust, or to propose and evaluate a new algorithm for extracting dense graphs from DNS graphs.

- To investigate sophisticated machine learning algorithms, such as semi-supervised learning algorithms, that required less training data and produce higher accuracy.

- To further study a technique to build and analyze a DNS traffic on-the-fly.

- In our study, we did not consider the problem with the queries issued by internal recursive DNS servers, or Network Address Translation (NAT) servers in the examined network. These queries may cause the incorrect mapping in the DNS failure graph because multiple queries are only mapped to IP addresses of the recursive DNS servers or NAT servers. Thus, it is interesting to study these open problems to get a more correct picture of the network.

In summary, this thesis leverages the use of DNS analysis in detecting botnets in particular and detecting malware in general. The results of this thesis confirms that DNS analysis can complement other network-based techniques to produce good malware detection results.

Bibliography

[1] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034 (Standard), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt

[2] ——, "Domain names - implementation and specification," RFC 1035 (Standard), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[3] K. Scarfone and P. Mell, "Guide To Intrusion Detection and Prevention Systems (IDPS), Sp-800-94," *Recommendations of the NIST National Institute of Standards and Technology (NIST)*, 2007.

[4] OpenDNS, "DNSCrypt," 2012. [Online]. Available: http://www.opendns.com/technology/dnscrypt/

[5] D. Eastlake 3rd, "Domain Name System Security Extensions," RFC 2535 (Proposed Standard), Internet Engineering Task Force, Mar. 1999, obsoleted by RFCs 4033, 4034, 4035, updated by RFCs 2931, 3007, 3008, 3090, 3226, 3445, 3597, 3655, 3658, 3755, 3757, 3845. [Online]. Available: http://www.ietf.org/rfc/rfc2535.txt

[6] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon, "Detecting malware domains at the upper DNS hierarchy," in *Proceedings of the 20th USENIX Security Symposium, USENIX Security*, vol. 11, 2011, pp. 27–27.

[7] D. Dagon, G. Gu, C. Lee, and W. Lee, "A Taxonomy of Botnet Structures," *Computer Security Applications Conference, Annual*, pp. 325–339, 2007.

[8] Trend, "Taxonomy of botnet threats," Trend Micro White Paper, Nov. 2006.

[9] S. Golovanov and I. Soumenkov, "TDL4 – Top Bot," Kaspersky Lab Analysis, 2011. [Online]. Available: http://www.securelist.com/en/analysis/204792180/TDL4_Top_Bot#2

[10] K. Chiang and L. Lloyd, "A case study of the Rustock rootkit and spam bot," in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets.* USENIX Association, 2007, p. 10.

[11] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage, "Spamalytics: An empirical analysis of spam marketing conversion," in *Proceedings of the 15th ACM conference on Computer and communications security.* ACM, 2008, pp. 3–14.

[12] D. Suarez, *Daemon.* EP Dutton, 2009.

[13] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: a case study on Storm worm," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. USENIX Association, 2008, p. 9.

[14] L. Gasster, "GNSO Issues Report on Fast Fluxing Hosting," ICANN, 2008. [Online]. Available: http://gnso.icann.org/issues/fast-flux-hosting/gnso-issues-report-fast-flux-25mar08.pdf

[15] W. Salusky and R. Danford, "Know your enemy: Fast-flux service networks," *The Honeynet Project*, pp. 1–24, 2007. [Online]. Available: http://www.honeynet.org/book/export/html/130

[16] "DNS-BH - Malware Domain Blocklist," 2012. [Online]. Available: http://www.malwaredomains.com/

[17] P. Royal, "Analysis of the Kraken Botnet," Damballa, Tech. Rep., 2008.

[18] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Proceedings of the 10th annual conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 48–61. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879148

[19] N. Jiang, J. Cao, Y. Jin, L. E. Li, and Z. L. Zhang, "Identifying Suspicious Activities through DNS Failure Graph Analysis," Technical report, University of Minnesota, Tech. Rep., 2010.

[20] C. Xiang, F. Binxing, Y. Lihua, L. Xiaoyi, and Z. Tianning, "Andbot: towards advanced mobile botnets," in *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*. USENIX Association, 2011, p. 11.

[21] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *Proceedings of the Network &amp; Distributed System Security Symposium*, 2008.

[22] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting malicious flux service networks through passive analysis of recursive DNS traces," in *Computer Security Applications Conference, 2009. ACSAC'09. Annual*. IEEE, 2009, pp. 311–320.

[23] Y. Jin, E. Sharafuddin, and Z. L. Zhang, "Unveiling core network-wide communication patterns through application traffic activity graph decomposition," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. ACM, 2009, pp. 49–60.

[24] C. Ding, T. Li, W. Peng, and H. Park, "Orthogonal nonnegative matrix t-factorizations for clustering," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '06. New York, NY, USA: ACM, 2006, pp. 126–135. [Online]. Available: http://doi.acm.org/10.1145/1150402.1150420

[25] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, “Building a dynamic reputation system for DNS,” in *19th Usenix Security Symposium*, 2010.

[26] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, “EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis,” in *Proceedings of NDSS*, 2011.

[27] M. Knysz, X. Hu, and K. Shin, “Good guys vs. Bot Guise: Mimicry attacks against fast-flux detection systems,” in *INFOCOM, 2011 Proceedings IEEE.* IEEE, 2011, pp. 1844–1852.

[28] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi, “Fluxor: detecting and monitoring fast-flux service networks,” *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 186–206, 2008.

[29] X. Hu, M. Knysz, and K. Shin, “Rb-seeker: auto-detection of redirection botnets,” in *ISOC Network and Distributed System Security Symp*, 2009.

[30] S. Huang, C. Mao, and H. Lee, “Fast-flux service network detection based on spatial snapshot mechanism for delay-free detection,” in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security.* ACM, 2010, pp. 101–111.

[31] X. Hu, M. Knysz, and K. Shin, “Measurement and analysis of global ip-usage patterns of fast-flux botnets,” in *INFOCOM, 2011 Proceedings IEEE.* IEEE, 2011, pp. 2633–2641.

[32] S. Hao, N. Feamster, and R. Pandrangi, “Monitoring the Initial DNS Behavior of Malicious Domains,” in *ACM SIGCOMM Internet Measurement Conference*, Berlin, Germany, November 2011.

[33] J. Nazario and T. Holz, “As the net churns: Fast-flux botnet observations,” in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on.* IEEE, 2008, pp. 24–31.

[34] J. Postel, “Internet Protocol,” RFC 791 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFC 1349. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[35] S. Deering and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 2460 (Draft Standard), Internet Engineering Task Force, Dec. 1998, updated by RFCs 5095, 5722, 5871. [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[36] J. Ellson, E. Gansner, L. Koutsofios, S. North, and G. Woodhull, “Graphviz—open source graph drawing tools,” in *Graph Drawing.* Springer, 2002, pp. 594–597. [Online]. Available: http://www.graphviz.org/

[37] Alexa Internet, “Alexa Top Sites,” 2012. [Online]. Available: http://www.alexa.com/topsites

[38] Wireshark, “Tshark-The Wireshark Network Analyzer,” 2012. [Online]. Available: http://www.wireshark.org/docs/man-pages/tshark.html

[39] S. McCanne, C. Leres, and V. Jacobson, “Tcpdump and Libpcap,” 2012. [Online]. Available: http://www.tcpdump.org/

[40] J. Xu, J. Fan, M. Ammar, S. Moon, Q. Zhang, J. Wang, and X. Li, “On the design of fast prefix-preserving IP address anonymization scheme,” in *Proceedings of the 9th international conference on Information and communications security*, Springer-Verlag. ACM, 2007, pp. 177–188. [Online]. Available: http://dl.acm.org/citation.cfm?id=505234

[41] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon, “Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme,” *Computer Networks*, vol. 46, no. 2, pp. 253–272, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128604001197

[42] Y. Rekhter and T. Li, “An Architecture for IP Address Allocation with CIDR,” RFC 1518 (Historic), Internet Engineering Task Force, Sep. 1993. [Online]. Available: http://www.ietf.org/rfc/rfc1518.txt

[43] D. Plonka and P. Barford, “Context-aware clustering of DNS query traffic,” in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, 2008, pp. 217–230.

[44] Spamhaus Project, “Spamhaus Domain Block List,” 2012. [Online]. Available: http://www.spamhaus.org/dbl/

[45] IANA, “Root Zone Database,” 2012. [Online]. Available: http://www.iana.org/domains/root/db/

[46] Python Software Foundation, “Python Programming Language,” 2012. [Online]. Available: http://python.org/

[47] T. Li and C. Ding, “The relationships among various nonnegative matrix factorization methods for clustering,” in *Data Mining, 2006. ICDM’06. Sixth International Conference on*. IEEE, 2006, pp. 362–371.

[48] J. Yoo and S. Choi, “Orthogonal nonnegative matrix tri-factorization for co-clustering: Multiplicative updates on Stiefel manifolds,” *Information Processing & Management*, 2010.

[49] C. Boutsidis and E. Gallopoulos, “SVD based initialization: A head start for nonnegative matrix factorization,” *Pattern Recognition*, vol. 41, no. 4, pp. 1350–1362, 2008.

[50] Cymru, “IP to ASN Mapping,” 2012. [Online]. Available: http://www.team-cymru.org/Services/ip-to-asn.html

[51] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond Blacklists : Learning to Detect Malicious Web Sites from Suspicious URLs,” *World Wide Web Internet And Web Information Systems*, 2009.

[52] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. USENIX Association, 2007, pp. 7–7.

[53] P. Tan, M. Steinbach, V. Kumar *et al.*, *Introduction to data mining*. Pearson Addison Wesley Boston, 2006.

[54] M. Mørup and M. N. Schmidt, "02450Toolbox," course 02450-Introduction to Machine Learning and Data Modeling, Mar. 2012. [Online]. Available: http://www.imm.dtu.dk/courses/02450

[55] P. Porras, H. Saïdi, and V. Yegneswaran, "A foray into Confickers logic and rendezvous points," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[56] S. Shin, G. Gu, N. Reddy, and C. Lee, "A large-scale empirical study of conficker," *Information Forensics and Security, IEEE Transactions on*, no. 99, pp. 1–1, 2011.

[57] "Malware Domain List," 2012. [Online]. Available: http://www.malwaredomainlist.com/

[58] "Zeus Tracker," 2012. [Online]. Available: https://zeustracker.abuse.ch/

[59] DNS-BH, "Conficker's Domain List," 2012. [Online]. Available: http://www.cert.at/static/conficker/all_domains.txt

[60] "Spam Archive," 2012. [Online]. Available: http://untroubled.org/spam/

[61] OpenDNS, "PhishTank," 2012. [Online]. Available: http://www.phishtank.com/

[62] Symantec, "Norton Safe Web," 2012. [Online]. Available: http://safeweb.norton.com/

[63] McAfee, "SiteAdvisor," 2012. [Online]. Available: http://www.siteadvisor.com/

[64] "Web of Trust," 2012. [Online]. Available: http://www.mywot.com/

[65] Internet Systems Consortium, "DNSDB," 2012. [Online]. Available: https://dnsdb.isc.org/

Appendix A

Parameters for Analysis Procedure

Some important parameters of the analysis procedure are described as the following. This appendix can be used as the references to tune the analysis procedure. The values of these parameters are set based on the observations on the data, hence they should be adjusted accordingly in order to obtain the best detection results.

Table A.1 summarizes the information of such parameters. The details on these parameters are described in the following paragraph.

Table A.1: Parameters for analysis procedure

Parameter	**Default**	**Var./Param.**	**Source file**
Connected components threshold	10	`mingsize`	`main.py`
Decomposition threshold	30	`clusterthres`	`main.py`
Density threshold	0.5	`dmin`	`main.py`
Maximum iterations	3000	`opt.iter`	`othnmf.m`
Malicious threshold	0.5	`--threshold`	`main.py`

(1) Connected component threshold (default `10`): the minimum number of vertices of interested connected components in DNS graphs. This parameter can be adjusted by changing the value of the global variable `mingsize` in the main program `main.py`.

(2) Decomposition threshold (default `30`): the minimum number of vertices of a connected component that need to be decomposed. The default value is set to `30` based on our expectation of having subgraphs with reasonable number of vertices due to our interest in detecting fast-flux and domain-flux. This parameter can be adjusted by changing the value of the global variable `clusterthres` in the main program `main.py`.

(3) Density threshold (default `0.5`): this value indicates the minimum density of the extracted subgraphs in the Graph Decomposition Module. This parameter can be changed by setting value of the global variable `dmin` in the main program `main.py`.

(4) Maximum iterations (default `3000`): the maximum number of iterations of NMTF algorithm. This parameter can be adjusted in the `matlab/orthnmf.m` script. Note that set this parameter to a large value can help the algorithm to converge and produce the optimal clustering, but also increase the running time of the algorithm.

(5) Malicious threshold (default `0.5`): the threshold to raise a warning about a malicious subgraph. In our Regression Function Module, the maliciousness scores produced are continuous in the range from 0 to 1. However, in practice we need to set a threshold to raise a warning about a malicious subgraph. This parameter can be set by the option `--threshold` of the main program `main.py`.

For information on other parameters and options, refer to the file `README.html` in the prototype package.

TRITA-ICT-EX-2012:53

www.kth.se