Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Bachelor's thesis

P2P Botnet Detection in Computer Networks

Jan Neužil

Supervisor: Ing. Tomáš Čejka

15th May 2014

Acknowledgements

I would like to thank my supervisor Tomáš Čejka and CESNET organization for leading my steps during the writing of this thesis. Mostly, my gratitude belongs to my family and friends for their support and patience during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on 15th May 2014 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Neužil, Jan. *P2P Botnet Detection in Computer Networks*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2014.

Abstract

This bachelor's thesis examines the issue of P2P botnets which is one of the major Internet security threats. The goal of this thesis is analysis and realization of a functional generic module for monitoring system which detects P2P botnet communication. The thesis focuses on graph-based techniques of detection which depends only on the basic communication flow information.

During the work on this thesis, I have created a functional module for detection of P2P botnet communication using the approaches of the mutual contacts graph and the dye-pumping algorithm. The real network traffic has been merged with synthetic P2P botnet topology. This dataset has been captured within a malware capture facility project on the university network. The implemented module achieves flawless results within the botnet dataset containing no false positives.

Keywords P2P botnet detection, network security, network traffic monitoring

Abstrakt

Tato bakalářská práce zkoumá problematiku P2P botnetů, které tvoří jednu z hlavních hrozeb na Internetu. Cílem této práce je analýza a realizace obecného funkčního modulu pro monitorovací systém detekující komunikaci v rámci P2P botnetu. Tato práce se zaměřuje na techniky detekce používající grafové struktury, které závisí pouze na základních informacích komunikačního toku.

Během práce na tomto projektu jsem vytvořil funkční modul detekující komunikaci v rámci P2P botnetu. Tento modul využívá grafu společných kontaktů a s ním spojený algoritmus. Reálná data ze síťového provozu byla sloučena s uměle vytvořenou topologií P2P botnetu. Tato data byla zachyceny na univerzitní síti v rámci jiného projektu na odhalování škodlivé síťové komunikace. Na těchto datech implemetovaný modul dosáhl bezchybných výsledků.

Klíčová slova detekce P2P botnetu, síťová bezpečnost, monitorování provozu v síti

Contents

List of Figures

List of Tables

Introduction

Botnets represent one of the biggest Internet security threats. Botnet is a set of infected computers gathered into one interconnected controllable structure. Botnet is operated via C&C (Command and Control) servers which maintain and control the infected computers through various channels. An infected computer is usually referred as a bot. Among various types of malicious activities, botnets are predominantly used for DDoS (Distributed Denial of Service) attacks, sending spam e-mails, setting up phishing sites, stealing sensitive personal data causing an identify fraud or brute-force cracking of hashed passwords [30].

Early botnets had used a centralized structure where all bots were operated from one or more C&C servers. This approach had provided a great vulnerability for owners of botnet (botmasters). This single point of failure could free whole botnet from its botmaster. These traditional botnets mostly have used IRC (Internet Relay Chat), HTTP (Hypertext Transfer Protocol) or other instant messaging protocols for communication, thus it was easier to trace them using several detection methods [19]. Many botnets were shutdown after they had been detected. Despite all the efforts of attackers to hide their commands via encrypted messages or different protocols, sooner or later a botnet has been discovered using different detection methods. With a single point of failure it had been only a matter of time when the botmaster was cut off from its bots.

Nowadays, botnets rely on decentralized P2P (Peer-to-Peer) structure which is much more resilient against single points of failure. Moreover, P2P networks are highly interconnected for reasons of reliability, which makes them distinguishable from client-server traffic [27]. However, distinguishing the P2P botnet C&C communication traffic from common P2P communication is much more difficult. Attackers also hide the communication with bots using encryption or randomly generated ports, thus most of the known centralized botnet detection techniques fail. Fortunately, one aspect can not be forged, that is the basic communication flow. Because botnet evolved into resilient P2P

decentralized structure and traditional botnet detectors can not discover sophisticated C&C communication, the need for a generic method to detect P2P botnet traffic in computer networks has been proposed.

In this bachelor's thesis, I will describe the issue of botnets in computer networks. Then, I will discuss and analyze existing botnet detection techniques. In the next part of the thesis, I will implement a module using generic method to detect P2P botnet communication. In the evaluation part, I will test the module on simulated network data. In the last part, I will conclude the work.

Goals of the Thesis

One of the goals of my bachelor's thesis is to study behaviour of P2P networks using flow data from monitoring system. The main goal is to implement a module which uses suitable graph structures to detect suspicious hosts which might have been compromised and now act as bots.

Some of the P2P detection methods rely on finding botnet-specific signatures, catching anomalies in traffic or examining payload, the main advantage of my module is generality based on the approach in [13, 27]. It uses only source and destination IP addresses and direction of flow as the essential information.

The result of my work is a functional module to detect suspicious behaviour in monitored networks. The module will be added to distributed modular system [8] among other detection modules and useful monitoring programs. My personal goal is to optimize the module to be able to detect suspicious in with very low false positive rate.

List of Module Requirements

Botnets are one of the major issues that appear on many current networks. To decrease malicious activity coming from botnets, further researches and approaches have to be proposed in the P2P botnet domain. This lack of generic efficient detection methods has been my primary motivation. I have attempted creation of a module which would detect P2P botnet communication in Czech Network Research and Education Network (NREN). Hence, I have defined these requirements to the work:

- independence to a P2P botnet kind,
- efficient detection using captured traffic in the short time period,
- implementation as a module to the distributed modular framework,
- usage of efficient graph algorithms and structures,

- successful test with low positive rate both on simulated and real network traffic.

Some of these requirements have also become tasks of my bachelor's thesis.

Text Structure of the Thesis

This bachelor's thesis is divided into four chapters:

Chapter 1 - Botnet Phenomenon
This chapter gives a brief overview of botnet phenomenon. It analyzes and describes botnet characteristics. This includes the way how user's computer becomes a bot, former and current solutions of botnet detection and overall brief to the given issue. The overview also involves traditional centralized botnet structure, the diversification of detection methods and other information gained in the exploration of facts about botnets.

Chapter 2 - Analysis and Design
This chapter describes the most suitable generic solution for the module. It formally defines the chosen approach using graph-based method, describes the module environment and gives a pseudo-code of the used algorithm.

Chapter 3 - Realization
This chapter describes the realization of the module. It defines structures and functions used in the implementation of P2P botnet detection module, describes all its parameters and interfaces and shows the module workflow.

Chapter 4 - Testing and Evaluation
This chapter shows results of testing. It tests the source code with dynamic analysis, describes the used dataset for the module evaluation and shows the result of detection on the simulated network traffic data.

Chapter 1

Botnet Phenomenon

Botnets can be divided into two major groups [30]. Traditional botnets have a centralized structure as it is shown in Figure 1.1. More resilient modern P2P botnets have a decentralized structure as it is shown in Figure 1.2. In this chapter, it will be briefly described detection techniques in centralized botnets as well.

To understand botnet phenomenon, I will describe the botnet characteristics. Then, I will describe a life-cycle of a botnet, how the host gets infected and when it is possible to detect malicious activity on a computer. In the end, I will summarize types of attacks and detection techniques. For this survey of botnet characteristics, I will cite and use article by Feily *et al.* [16].

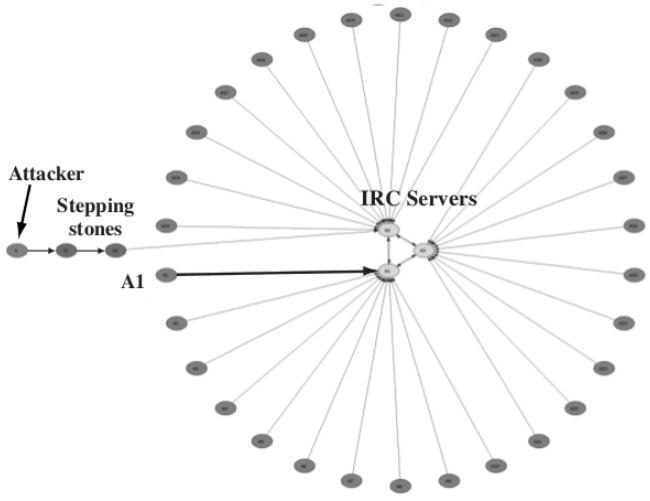

Figure 1.1: Centralized topology of a botnet (source: Ditrich *et al.* [15])

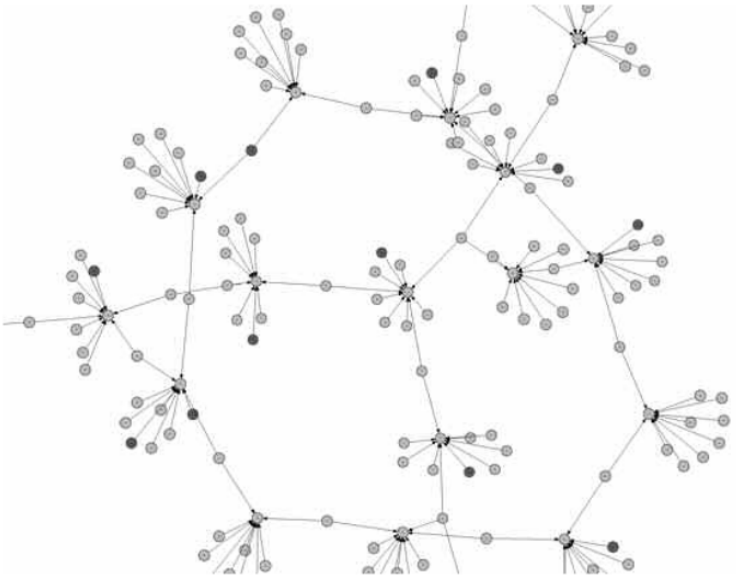

Figure 1.2: Decentralized topology of a botnet (source: Ditrich *et al.* [15])

1.1 Botnet Characteristics

The word **botnet** is a combination of the words *robot* and *network*. Administrators of botnets called **botmasters** exploit vulnerabilities of unsecured hosts to take control over a targeted computer. When a targeted computer is infected with a malicious application, the computer becomes a servant to its botmaster, a **bot**.

The application can also reproduce itself using self-propagating virus. Mostly, it spreads the infection when the victim is connected to the Internet. Methods of infections are similar to common malware intrusion such as trojan horses, phishing, spam e-mails or social engineering techniques to download malicious bot code. There are plenty of methods to get infected and the attackers tend to be very resourceful.

After all, botnet business is very lucrative. Today, a botnet can be illegaly rented for malicious activity at customer's wish. According to iDefense team from Verisign [2], the average prize 9$ has been calculated per hour of botnet services in 2010. Most of the offers have been found on the Russian blogs.

Very significant characteristic of a botnet is a communication. They use a C&C communication channels to control and to keep bots in active state. A multi-tier architecture of C&C channels provides anonymity to the botmaster. Many communication protocols and a wide range of logical network topologies are used to avoid being detected. Because of that, botnets are usually classified by their communication protocols.

According to the communication protocol classification, major groups are formed by IRC-based, HTTP-based, DNS-based (Domain Name System) and P2P-based botnets. Most of the traditional centralized botnets used IRC protocol as it has been originally designed for communication among large number of hosts in social chat rooms. IRC protocol has been popular due to its scalability, flexibility and also modifiability. IRC is a open protocol which allow botmasters to modify and customize it to their demands. P2P-based botnet uses recent P2P protocols which makes them more resilient against single point of failure or takeover [31].

1.2 Botnet Life-cycle

According to Feily *et al.* [16] a typical botnet life-cycle can be described in five stages: initial infection, secondary injection, connection, malicious command and control, update and maintenance. These phases are described in the list below:

Phase 1: The attacker finds and exploits target's system vulnerabilities to gain access to victim's machine. Once a target's weakness is found, the attacker uses different infection techniques to proceed to the second phase of injection.

Phase 2: The attacker runs script or binary program which downloads control interface for the botmaster from specific location usually via FTP (File Transfer Protocol), HTTP or P2P protocols. When the control interface is downloaded and installed on victim's computer, the machine becomes a part of a botnet. The bot application starts automatically every time the computers reboots.

Phase 3: The bot establishes a reliable C&C channel with its botmaster. When a bot is connected to the Internet, it will take commands from botmaster to perform several malicious activities, described in the next section.

Phase 4: The bot performs malicious activity. During this phase of the botnet life-cycle, a user can notice unusual behaviour of the infected machine as the computer consumes too much CPU (Central Processing Unit) time or the computer sends too many packets without a reason.

Phase 5: The infected host downloads and updates bot application. Botmasters maintain their army of bots and try to avoid detection by switching protocols or performing server migration.

1.3 Types of Attacks

In this section, I will provide a quick overview of the attack types a botnet might perform. There are several serious threats a botnet might do. Most of them are listed in Table 1.1.

Table 1.1: Malware Threats

Threat	Description
Spam	Sending malware or spam via e-mails
Dictionary attack	Guessing user's password to gain access
DDoS	Distributed denial of service of selected servers
Identity fraud	Stealing user's identity for illegal activities
Distributed computing	Using CPU's performance to decrypt passwords
MiM attacks	Man in the middle attacks to spread bot infection
Network scanning	Attacking random addresses or scanning for open ports

1.4 Centralized Botnets Detection

Even though botnets represent a distinctive threat to the network security, there are not many formal solutions to this problem. Most of the common users have never heard about botnets. Recently, there were massive cyber-attacks on US government websites after the biggest file sharing server had been shutdown. Some of the common users might unconsciously had taken a part in the attack as their computer sent massive amount of request to the government servers. This is the only chance when the common user can notice computer's suspicious activity, because when a user's computer gets infected, it is hard, almost impossible, to seek and destroy bot application without reinstalling an operating system. A protection of end devices against botnets consists of genuine and updated operating system, anti-virus software and properly secured network.

From the client side there is almost nothing a user can do to detect bot application. Mostly, we place botnet detection into an ISP environment. We can split a botnet detection into two significant groups. Active and passive. Active solutions use honeypots [25] which work as a bait for the attackers. The honeypot framework acts as a legitimate server or host with all features. When an attacker scans network, he might encounter a honeypot. Using the honeypot framework, botnet behaviour can be observed and studied to improve detection techniques or to restrict botnet's C&C server access to the Internet. Although honeypots seem a reasonable solution, more common practice are passive methods of detection.

In the survey of botnet detection techniques Feily *et al.* [16] distinguish passive methods as signature-based, anomaly-base, DNS-based and mining-based.

1.4.1 Signature-based Detection

Signature-based technique is very useful when a defender has knowledge of botnet behaviour and signatures. It can be configured with a set of rules or signatures to capture traffic which is considered suspicious. There are several intrusion detection systems (IDS), but the most widely known is Snort [4]. It is written as open source in the C language which makes it a perfect free traffic monitoring tool for ISPs. Although signature-based method is very popular, it only allows to detect known botnets based on their behaviour. For unknown botnets different techniques had to be developed.

1.4.2 Anomaly-based Detection

Another technique based on capturing anomalies in network traffic such as high volumes of traffic, traffic on unusual ports, unusual system behaviour or high network latency is often used in network monitoring software. These anomalies in traffic could indicate potential unknown botnet traffic and presence of malicious bots. Even though anomaly-based detection solves the issue with unknown botnets, problem might occur with botnets which have not been used for attacks yet, hence no anomalies can be detected in traffic.

In 2006, Binkley *et al.* [9] proposed an algorithm that combines IRC tokenization with TCP-based anomaly detection. With the aid of IRC message statistics they have created an effective system that detected client botnets. However, the proposed algorithm could be have been easily bypassed using simple encryption in IRC messages.

Suspicious encrypted traffic can be also detected based on flow data in transport layer as it was proposed in [20]. Different anomaly-based detection technique called Botsniffer [19] explores C&C channels in local area network. It focuses on synchronization events and activities of bots. Comparison of all detection techniques can be found in Table 1.2.

1.4.3 DNS-based Detection

DNS-based technique is similar to anomaly-based detection as it looks for DNS traffic anomalies. When bots initiate a connection with C&C server to get commands they perform DNS queries to locate and get IP address of the C&C server. Dynamic DNS (DDNS) service allows that C&C servers can often change location. In case that authorities block a C&C server at a certain IP address, the botmaster can easily set up another C&C server instance changing only the IP address. Hence, several methods to detect botnets from DNS traffic have been proposed.

One of the technique relies on a mechanism to detect C&C servers based on domain names with abnormally high or temporally concentrated DDNS queries rate [14]. Similar approach is used in [21], but both of the techniques can be confused by generating fake DNS queries. Another approach was based

on abnormally recurring NXDOMAIN reply rates proposed in 2006 [28], which led to less false positives events.

A promising technique was proposed in [26] which analysis DNSBL (DNS-based Black-hole List) lookup traffic. Botmasters themselves perform lookup against DNSBL to determine bot blacklist status. Unfortunately, distinguishing botnet DNSBL queries from legitimate DNSBL lookup traffic leads to higher false positives. Last DNS-based approach mentioned in this thesis monitors group activity anomalies in DNS traffic [11]. This robust detection technique is very effective regardless botnet type, on the other hand it requires too much processing time.

1.4.4 Mining-based Detection

Since most of the anomalies occur in the phase of performing malicious activities such DDoS attacks, it is very difficult to monitor C&C botnet traffic in the passive stage of bots. For these reasons, some of data mining techniques have been developed to efficiently detect botnets, e.g. classification, clustering or machine learning.

In [32] authors use machine learning technique based on flow characteristic, but this approach requires access to payload which can be also encrypted. Furthermore, it leads to higher false positive rate because of botnet diversity. In [23] authors introduce multiple log correlation for C&C traffic detection. This robust and effective solution is based on classification of an entire flow to identify the botnet C&C traffic. It also does not require an access to payload content. In [18] authors introduce Botminer which clusters similar communication traffic and similar malicious traffic. Then, it performs cross cluster correlation to identify bots. This robust solution is protocol independent and provides very low false positive rate.

1.4.5 Comparison of Botnet Detection Techniques

Most of the current botnet detection techniques work only on specific botnet C&C communication protocols and structures. As botnets change their C&C communication architecture, these methods might become ineffective. In summary, because of botnet structure and protocol diversity, botnet detection becomes a very challenging task. In the Table 1.2, different detection techniques can be compared as they have been mentioned in paragraphs above.

1.5 P2P Botnets Detection

In the section above, basic botnet characteristics and detection method have been explained. Most of the proposed methods were developed in the first decade of the twenty-first century which had been predominantly an era of

Table 1.2: Brief comparison of botnet detection techniques (source: Feily *et al.* [16]). The table contains seven columns denoted by a letter abbreviation. *T* stands for detection technique, *D* for detection approach proposed in cited article, *U* for unknown bot detection (generic approach), *P* for protocol independence, *E* for encrypted bot detection, *R* for real-time detection and *L* for low positive rate.

Technique (T)	D	U	P	E	R	L
Signature-based	[4]	No	No	No	No	No
Anomaly-based	[9]	Yes	No	No	No	No
	[20]	Yes	No	Yes	No	Yes
	[19]	Yes	No	Yes	No	Yes
DNS-based	[14]	Yes	No	Yes	No	No
	[21]	Yes	No	Yes	No	No
	[28]	Yes	No	Yes	No	Yes
	[26]	Yes	No	Yes	Yes	No
	[11]	Yes	Yes	Yes	No	Yes
Mining-based	[32]	Yes	No	No	No	No
	[23]	Yes	Yes	Yes	No	Yes
	[18]	Yes	Yes	Yes	No	Yes

traditional centralized botnets. As majority of these botnets had been shutdown, attackers found a way to make their C&C communication almost invisible using P2P network topology. In this section, I will focus only on botnets of a new era which use the P2P network topology. Also, I will describe different detection techniques and analyze the chosen approach to the desired module.

1.5.1 P2P Networks

A P2P network is a type of decentralized and distributed network architecture in which every individual host acts as both client and server. The hosts, also called "peers", supplies and consumes resources at the same time independently of central servers. Even though P2P networks had been described and used in many applications, the concept was massively popularized by the music-sharing application Napster [5]. This application was released in 1999 and it rapidly became popular as it has allowed to share music among its users for free. After two years, the company ran into legal difficulties over copyright infringement with music publishers.

P2P networks are implemented in very large scale of applications that common users use every day, *e.g.* BitTorrent protocol which is used for file sharing among peers, but also for the Internet piracy. Another example might be Skype [6] to carry voice and video over IP protocols, or deflationary digital currency like Bitcoin [7] which is really disputable these days.

1.5.2 P2P Detection Techniques

Although detection of P2P botnet is more difficult than a detection of centralized botnets with single point of failure, several approaches have been proposed. One of the solutions might be using a clustering based node behaviour profiling approach. It captures the node behaviour clusters in a network and uses formal statistical tests on popular behaviour clusters in this network [10]. Another approach presents a localization of botnet members based on the unique communication patterns arising from their overlay topologies used for command and control [24]. Also mining-based methods are used. One approach classifies network traffic behaviour during a given time interval using machine learning classification technique [33]. Another approach proposes a data mining scheme to capture anomalies from network traffic based on certain flow fields (e.g. packet size) [22]. All of these proposed techniques are robust and provide low false positive rate.

Chapter 2

Analysis and Design

In this chapter, I will describe the chosen approach for P2P botnet detection method which is the most suitable solution for the desired module. In 2010, Coskun *et al.* proposed a simple efficient method to detect P2P botnets using only flow records captured on a network border [13]. To confirm suggested method, in [27] authors have critically reviewed the proposed approach with the highest result of quality for the edge router monitoring. They also improve this approach by presenting a computationally less complex algorithm. Among all the other detection techniques, this fulfills all the needed requirements.

The proposed technique is based on the observation that peers of a P2P botnet communicate with other peers in order to receive commands and updates. P2P bots usually hide their C&C communication through the use of various of protocols and encrypted packets which is an effective way not to be discovered by mining-based or anomaly-based botnet detection methods. Bots usually select their peers randomly and independently (*i.e.* unstructured P2P topology). Nevertheless, one aspect can be hardly altered and that is a basic communication flow. This flow record information is the only requirement for graph-based botnet detection method which forms a **Traffic Dispersion Graph** (TDG). IP addresses represent nodes and flow records between two nodes within a certain monitored interval represent an edge. Such TDG can be created in a typical ISP environment on an edge router to the Internet as it is shown in Figure 2.1.

2.1 Mutual Contacts Graph

In [13] the mutual contact between two peers is defined as:

> Although different bots may communicate with different peers, we show that for P2P botnets with an unstructured topology, where bots randomly pick peers to communicate with, there is a surprisingly high probability that any given pair of P2P bots communi-

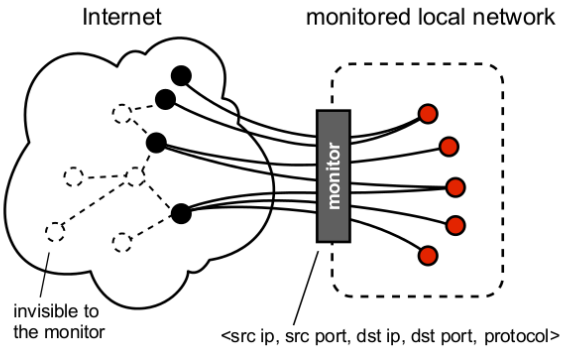

Figure 2.1: Monitored network (source: Ruehrup *et al.* [27])

cate with at least one common external bot during a given time window. In other words, there is a significant probability a pair of bots within a network have a **mutual contact**.

In order to create a Mutual Contacts Graph (MCG), these three steps must be followed:

1. TDG must be retrieved from collecting flow records in given time period. Nodes are represented captured hosts, both local and external, and edges represent the communication among local and external peers.

2. TDG is filtered. Since almost every peer communicates with popular servers such as search engines, social networks or news servers, a mutual contact would be created among these peers. Hence a privacy-threshold is set to solve this problem.

3. Final MCG is created by searching for mutual contacts of local hosts in TDG. Once a mutual contact is found, an edge between two local hosts will be weighted by number of mutual contacts among external peers.

All of these steps are illustrated in Figure 2.2. After a MCG is created, the technique computes bot membership probability using a dye-pumping algorithm which is described later in section 2.4.2. In order to detect suspicious local hosts, this algorithm requires a list of known seed bots to perform breadth-first search over the created MCG. As a result, a list of local bots, which are above given probability threshold, is returned.

Briefly, mutual contacts graph-based technique satisfy multiple requirements which are summarized in [13] by following list:

- The proposed method is not an anomaly detection scheme and hence does not require P2P bots to exhibit any overtly malicious activity.

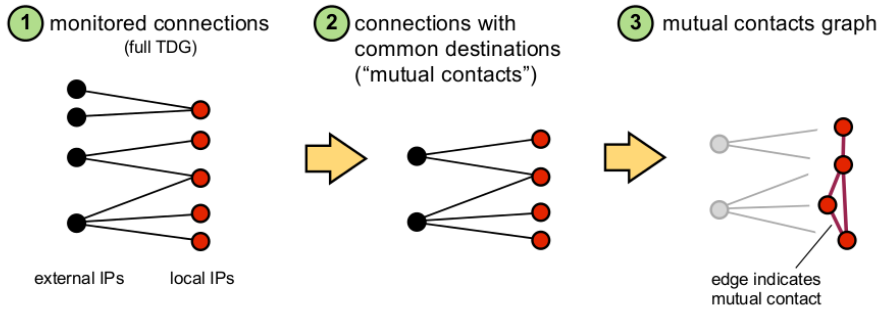

Figure 2.2: Creation of mutual contacts graph (source: Ruehrup *et al.* [27])

- Similarly, it is not a behaviour clustering algorithm and therefore does not require any common behaviour exhibited by all the bots.
- It utilizes the pairwise mutual contact relationships between pairs of bot peers, which arise due to P2P C&C communications.
- The proposed method is generic and does not depend on specific properties of specific botnets. Therefore, it does not require reverse engineering bot binaries or C&C protocols.
- Contrary to existing graph-based network traffic analysis methods, the proposed method does not require any access to backbone traffic. Mutual contact relationships are deduced locally at an edge router.

Mutual contacts graph-based technique is the most appropriate method which fulfills all the proposed requirements. I use mutual contacts graph-based technique as chosen approach for the implementation of the P2P botnet detection module to a distributed monitoring system.

2.2 Module Deployment

In this section, I will briefly describe the future deployment of the proposed P2P botnet detection module into the Nemea framework, the purpose of the framework and how it works. This description is derived and cited from technical report [8] about the Nemea framework.

In order to make Internet safe place, many organizations around the world try to avoid or detect malicious activity over their networks. CESNET [3] is the NREN operator in the Czech Republic. CESNET maintains and provides the network connection to many Czech educational institutions such as

universities, research centers, etc. It also connects the network with foreign countries via its network CESNET2.

Nemea is a framework developed in CESNET. It allows for an assembly of a system for automated analysis of flow records gathered from network monitoring processes in real time. The system consists of separate building blocks called modules. The modules are interconnected by interfaces. All interfaces are one-way only and they transfer data in the form of individual records (*e.g.* flow records or records describing detected anomalies). Each module is a standalone application, *i.e.* it runs as a separate system process. The modules in the Nemea framework process data stream-wise, *i.e.* data arrives to a module as individual records one by one [8].

The Nemea framework has been designed to detect network security threats in CESNET2. Common approaches monitor and store data using network probes and storage systems in a certain time period, thus the collected data are analyzed after the given time period. In contrast, the Nemea process data stream-wise without storing them on hard drives which creates a space for real-time detection. The goal of this thesis is to find an ideal approach of P2P botnet detection and implement it as a stream-wise module to the Nemea framework.

2.3 Dynamic Libraries

In this section, I will explain a traffic analysis platform (TRAP) and dynamic unified record (UniRec) which have been developed by CESNET for the Nemea framework. Both TRAP and UniRec are written in the C language as dynamic libraries which are linked to the module. In brief, TRAP serves as a communication interface between modules using TCP (Transport Control Protocol) or Unix socket and UniRec record is used as a specific flow data format which is sent over TRAP interfaces.

2.3.1 TRAP Library

In order to process data in stream-wise character, a shared object library called libtrap has been created. It allows to efficiently send flow records with very high throughput as vast volume of traffic flows in the network CESNET2. TRAP architecture in Fig 2.3 consist of four hierarchical layers. The lowest layer represents an interface type which may be TCP or Unix socket. A buffering sub-layer aggregates multiple transfer request, thus it reduces time-consuming operations. Higher sub-layer provides multiple components such as time-out feature to set blocking or non-blocking calls on interfaces, multi-read feature to receive flows on multiple interfaces in parallel or auto-flush features which allows to prematurely send buffer after a time-out has elapsed on interfaces. The highest layer of libtrap library is API (Application Programming Interface) to provide access to libtrap input and output interfaces in modules.

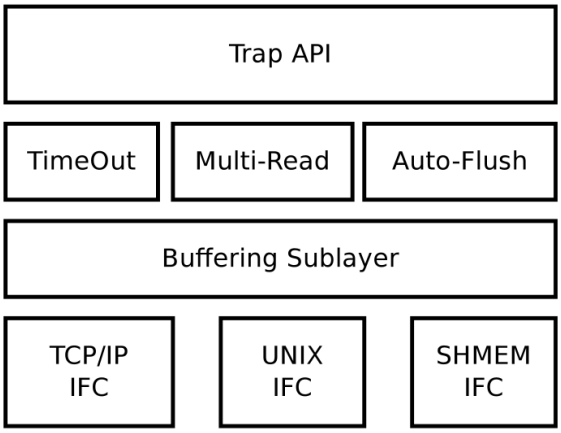

Figure 2.3: TRAP architecture (source: Bartoš *et al.* [8])

TRAP API enables rapid development of new Nemea modules and allows to set various parameters to achieve the best throughput performance.

2.3.2 UniRec Protocol

In the Internet a various types of attacks can be observed whether it is conducted by a botnet or by other way. To detect all malicious activities in the CESNET2 network, the Nemea consists of various detection modules. Each module requires a specific information to be sent on the interface. For this reason the UniRec has been developed. UniRec allows to send various types of data in a flexible format which can be dynamically created for the required purposes. Currently, basic flow information uses the IPFIX (IP Flow Information Export) format [12], which can be extended to contain more fields. In Figure 2.4 we see that the flow record is divided in static and dynamic part. In the static part, all the fields have static size. In the dynamic part, module-specific fields can be created.

2.4 Module Algorithms

In this section, I will provide a theoretical background of a MCG and dye-pumping algorithm used in the final module. I will be using algorithms and formulas proposed by Coskun *et al.* [13] and reviewed by Ruehrup *et al.* [27] confirming the functionality of the proposed technique. First, I will focus on explanation of constructing a MCG. In the second part, I will describe a dye-pumping algorithm.

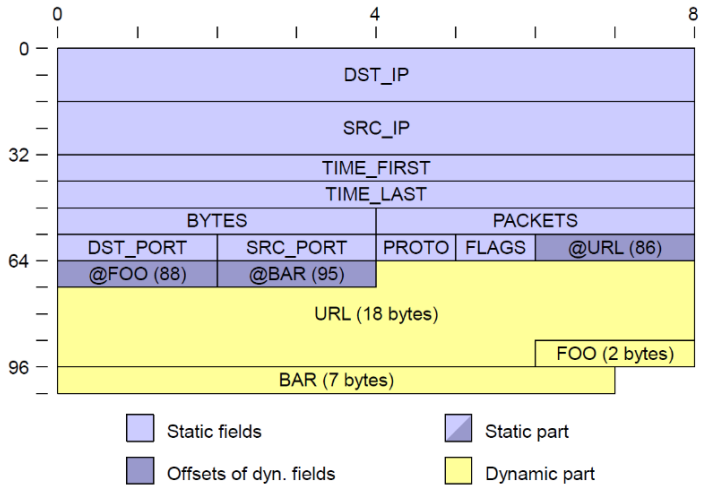

Figure 2.4: UniRec flow record (source: Bartoš *et al.* [8])

2.4.1 MCG

As a first step, TDG must be created from captured flow records. For better explanation of the MCG, I will use a simple scenario illustrated in Fig 2.5. In given scenario *Host A* shares one mutual contact with *Host B* since they both communicated with *Host X*. Similarly *Host B* is linked with *Host C* through *Host Y* and *Host Z*. If *Host A* is a known bot member, then *Host B* becomes suspicious as they have a mutual contact via *Host X*. Likewise, if *Host B* takes part in a botnet, *Host C* is likely to be member as well. Since *Host D* and *Host E* do not share a mutual contact with any suspicious host, they will be considered as benign hosts. This approach uses the fact that P2P botnet members receive commands from multiple peers.

It is the indisputable fact that next to P2P botnet traffic, legitimate traffic will generate mutual contacts as well since almost every host in the network communicates with popular sites such as social media or search engines. This causes that every host is connected to most of the other hosts via those popular mutual contacts. Therefore, if *Host X* has not communicated with almost anyone within a local network and *Host A* is a known bot, then with very high probability *Host B* is also a member of the same P2P botnet. Hence, only external hosts which have communicated with less than k local hosts are used to derive the MCG. Here, these contacts are referred as **private mutual contacts** and k is the **privacy-threshold**.

Formally, a MCG is defined as $MCG = (N, E)$ where N is set of local hosts and E is set of edges indicating a private mutual contact. Each edge has a capacity determined by the number of private mutual contacts between corresponding hosts. If w_{ij} represent a capacity of the edge between two nodes

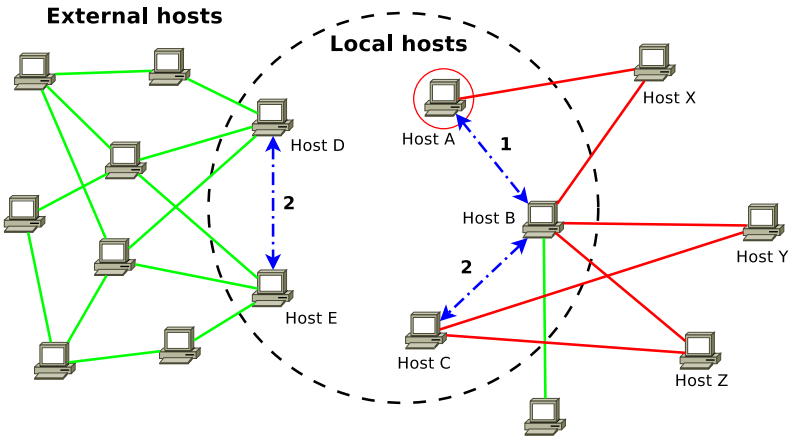

Figure 2.5: Illustration of P2P botnet communication and MCG. The local network contains one seed bot (Host A), two potential bots (Host B and C) and two benign hosts (Host D and E). The external network contains three potential bots (Host X, Y and Z) and other legitimate hosts. The green edges represent legitimate communication. The red edges represent suspicious communication. The blue lines are edges indicating mutual contact.

N_i and N_j, then:

$$w_{ij} = w_{ji} = |S(N_i) \cap S(N_j)|$$

where $S(N_i)$ represents the set of private mutual contacts. The cardinality of sets intersection determines the final capacity of the edge. If there is no private mutual contact between nodes, no edge is created, *i.e.* $w_{ij} = 0$.

2.4.2 Dye-pumping Algorithm

To detect suspicious hosts in local network one additional the knowledge of a seed bot is required. This information can be obtained from another detection techniques or blacklist of known bots. When the MCG is constructed, the designed algorithm dyes seed bot and spreads the dye over its mutual contacts. The algorithm is called the **dye-pumping algorithm** [13].

In real environment few benign bots would receive some amount of dye due to sharing of a mutual contact with P2P bots. This potentially result in false positives. Intuitively, there is a very low probability that suspicious P2P bots will be at a long distance from the seed bot. This unfavorable result can be eliminated by setting a suitable threshold of pumping depth. As it has been mentioned, the dye-pumping algorithm iteratively spreads dye over the MCG from the seed node and selects the hosts which accumulates more dye than a threshold. Original dye-pumping algorithm [13] uses the complete transition matrix of the MCG, but a computationally less complex algorithm has been

proposed in [27]. This algorithm uses a BFS (Breadth-First Search) technique to effectively spread dye from the seed bots to their neighbors.

If host shares a mutual contacts with many other hosts, it might indicate another P2P application. It can result in false positives. For this reason and for proportionately distributing the dye among neighbors, a **dye-attraction coefficient** is specified. It is denoted by γ_{ij}. It indicates what portion of the dye arriving at node N_i will be distributed to node N_j in the next iteration. It is computed as follows:

$$\gamma_{ij} = \frac{w_{ij}}{(D_j)^{\beta}}$$

where D_j is the degree of node N_j (*i.e.* number of edges that N_j has) and β is the **node degree sensitivity coefficient**. Thus, node with high degree receives less dye than host with low degree.

The pseudo-code of the dye-pumping algorithm is given in (1). This pseudo-code has been proposed in [27] and it it slightly modified. To understand the given algorithm, the explanation of variables is described in the following list:

- G_{MC} - mutual contacts graph
- B - set of seed bots
- C - current examined set
- T - next set to be examined
- V - visited set
- max_depth - maximum depth in graph from seed bots
- $\overline{w}$ - sum of edge weights leading from the current code
- a - dye-attenuation factor
- γ - dye-attraction coefficient
- d - dye-distribution factor (default: 0.5)

Algorithm 1 Dye-pumping algorithm

```
1: function FIND_BOTS(G_MC, B, max_depth)
2:     C ← B                          ▷ set of seed bots forms the initial current set
3:     V ← T ← ∅
4:     foreach i ∈ N (N ⊂ G_MC) do          ▷ set initial dye to all hosts
5:         dye[i] ← 0
6:     foreach i ∈ B do                     ▷ set initial dye to all seed bots
7:         dye[i] ← 1
8:     for n ← 1 to max_depth do            ▷ search graph to maximum depth
9:         if C ∈ ∅ then                    ▷ break loop if current set is empty
10:            break
11:        foreach n ∈ C do                 ▷ decrease dye, distribute later
12:            dye_old[i] ← dye[i]
13:            dye[i] ← dye[i] * (1 − d)
14:        foreach i ∈ C do
15:            w̄ ← 0
16:            foreach j ∈ S(i) do
17:                if j ∉ V then
18:                    a[j] ← (D_j)^γ
19:                    w̄ ← w̄ + w_ij * a[j]
20:            if w̄ = 0 then
21:                dye[i] ← dye[i]/(1 − d)
22:            foreach j ∈ S(i) do          ▷ distribute dye to neighbors
23:                if j ∉ V then
24:                    dye[j] ← dye[j] + dye_old[i] * d * w_ij * a[j]/w̄
25:                    if j ∉ C then
26:                        T ← T ∪ j
27:        foreach i ∈ T do                 ▷ remove hosts with not enough dye
28:            sum_dye ← 0
29:            foreach j ∈ S(i) do
30:                if j ∈ C then
31:                    sum_dye ← sum_dye + dye[j]
32:            if sum_dye < dye_threshold then
33:                T ← T \ {i}
34:        V ← V ∪ C
35:        C ← T
36:        T ← ∅
37:    return all i ∈ V where dye[i] > dye_threshold
```

Chapter 3

Realization

In this chapter, I will provide a description of parameters, functions, structures, algorithms and basic workflow used in the P2P botnet detection module. The module is written in the low-level C language that allows high performance and low resource consumption. As mentioned in the previous chapter, the module is designed to be a part of the Nemea framework. It uses TRAP interfaces to communicate with other modules. The module is run using CLI as no GUI is required for handling the process.

3.1 Module Parameters

In order to achieve flexibility, the module can be initialized with various parameters specifying behavior and thresholds during the module initialization. When changing the observation interval, the size of graph structure might vary, thus the TDG will contain different number of hosts. To preserve the module accuracy, privacy-threshold and minimum dye threshold have to be modified. The module also allows to modify other values such as maximum depth for the BFS algorithm, dye distribution factor or degree sensitivity coefficient. The module can also log the entire graph structure to a file based on a verbosity level, but in case of higher verbosity level combined with thousands of hosts, the log file might consume a large amount of space on a hard-drive. The list of all possible parameters in the manual page format is listed below:

-C COEF
Sets the degree sensitivity coefficient, 2 by default.

-D DEPTH
Sets the maximum depth of searching in the mutual contacts graph.

-F FAC
Set the dye distribution factor, 0.5 by default.

-h
Prints the help message.

-i IFC_SPEC
Mandatory parameter to specify the interfaces.

-L LEVEL
Prints graph structure based on given verbosity level, range 1 to 5.

-p NUM
Shows progress by printing a dot every N flows.

-P P
Sets the bot probability dye threshold, range 0 to 1.

-u TMPLT
Specifies UniRec template expected on the input interface for flows.

-s
Sets the flag to print flow statistics, for testing purposes only.

-T LIMIT
Sets the privacy-threshold to remove popular external hosts.

-v
Sets the TRAP verbosity level.

3.2 Module Interfaces

In this section, I will describe module communication with other modules using interfaces. The module has two input interfaces and one output interface. One input interface receives basic communication flow containing essential information in order to create TDG. This information consists of source and destination IP address, protocol and direction of flow using UniRec protocol. This information is used to create a TDG of communication. Another input interface receives the list of known bots or suspicious hosts with given probability of being part of a botnet. This information is used as the set of seed bots in dye-pumping algorithm. After the module finds all suspicious hosts above minimum dye threshold, this information is sent on the output interface to further evaluation. The illustration of module interfaces is shown in Figure 3.1.

3.3 Module Structures

In this section, I will describe specific structures used by the module. The structures help to store the information about parameters and communication graphs. The module uses five specific structures described below:

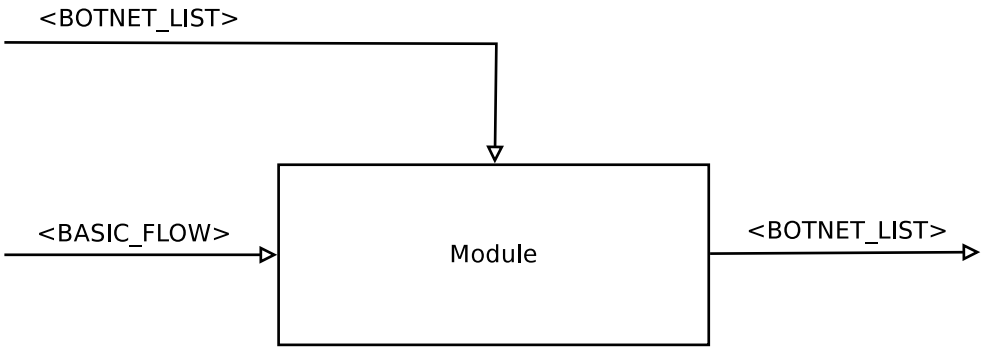

Figure 3.1: Module input interfaces receive dynamic *BASIC_FLOW* and static *BOTNET_LIST*. The *BASIC_FLOW* has to contain at least these UniRec fields: *SRC_IP*, *DST_IP*, *PROTOCOL* and *DIR_BIT_FIELD*. The *BOTNET_LIST* contains only *BOTNET_BOT_IP* and *BOTNET_PROBABILITY* fields. Then, the module finds other suspicious bots and sends the information in *BOTNET_LIST* format on the output interface.

params_t
: This structure keeps all parameters received during the module initialization. The list of parameters is provided in section 3.1. The structure is passed to functions whenever it is required.

host_loc_t
: This structure keeps all information about a local host. The structure contains the host IP address for identification, the host status for BFS algorithm, the host dye coefficient, number of accesses to the host, array of pointers to external hosts with whom the host has communicated and array of pointers to local hosts with whom the host shares mutual contact and array of weights of the respective mutual contacts edges.

host_ext_t
: This structure is similar to the local host structure, but it keeps only the host IP address, number of accesses to the host and array of pointers to local host with whom the the host has communicated.

node_t
: This auxiliary structure constructs a binary tree of hosts based by their IP addresses. The node consists of left and right pointer based on a single bit and the leaf contains a pointer to the host. The structure is illustrated in Figure 3.2. The purpose of using the binary tree is to effectively search arrays of hosts based on their IP addresses. The time complexity of this search is $\Theta(32)$ and $\Theta(128)$ for IPv4 and IPv6,

respectively. The binary tree is sparse, hence only required nodes are allocated.

graph_t

This structure is essential to most of the functions used in the module. It contains crucial information such as pointers to the root nodes of binary trees distinguished by IP version, array of pointers to all local hosts, array of pointers to all external hosts, array of pointers to suspicious local hosts and array of pointers to suspicious external hosts.

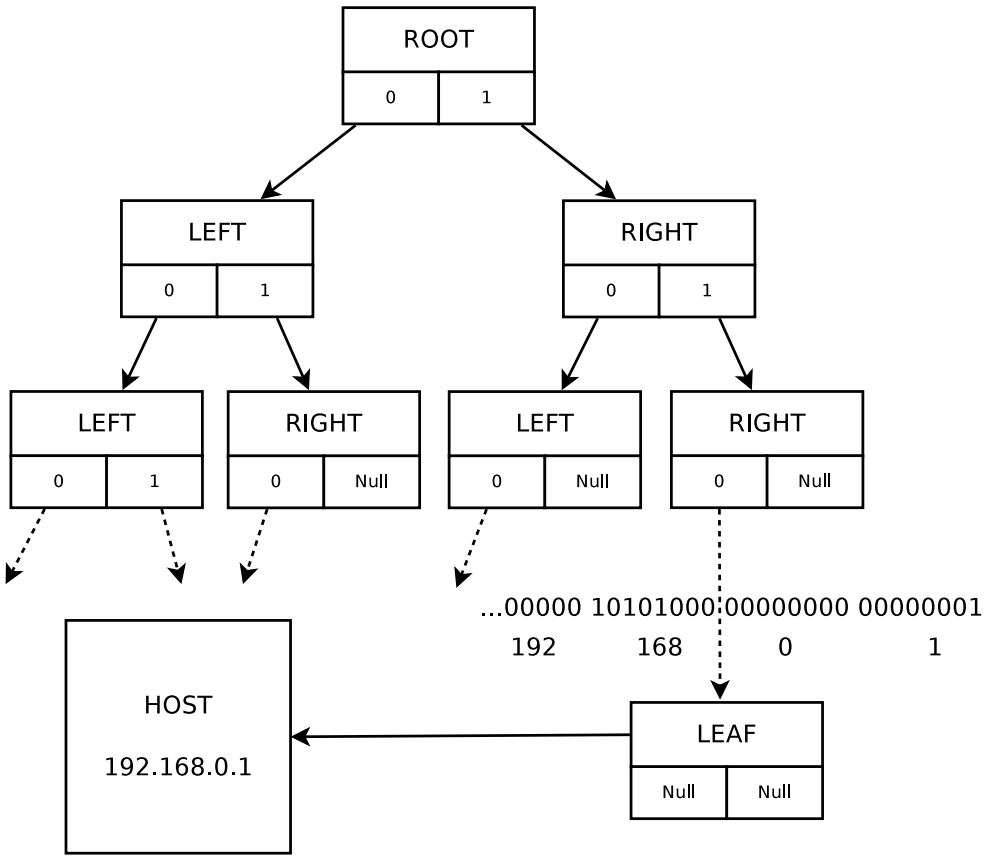

Figure 3.2: IP binary tree to effectively search among hosts based on a IPv4 or IPv6 address. The search function uses bits in IP address to get to the leaf which has a pointer to the host.

3.4 Module Workflow

In this section, I will describe basic module workflow and functions which are called during a runtime of the module. The principal handler of the module is the function *main()* which sequentially calls individual functions. It also controls all possible errors which might occur during the run of the program. The flow chart of the module workflow is illustrated in Figure 3.3. The sequence of called functions with brief description is written in the following list:

1. Function *module_init()* is called to prepare module structure containing information about number of interfaces needed for the module. In this case, the module requires two input interfaces and one output interface. Afterwards the TRAP interfaces are initialized using predefined macros.

2. Function *params_init()* is called to set all needed input parameters into single structure. The pointer to allocated structure is hand over to many other functions. The list of parameters in provided in section 3.1.

3. Function *get_hosts()* is called to receive all the traffic to be examined on the first input interface. The traffic is filtered to support only TCP and UDP (User Datagram Protocol) protocols. Based on the flow direction, the function distinguishes local and external host with appropriate IP address. Then, both local and external IP address are searched using the

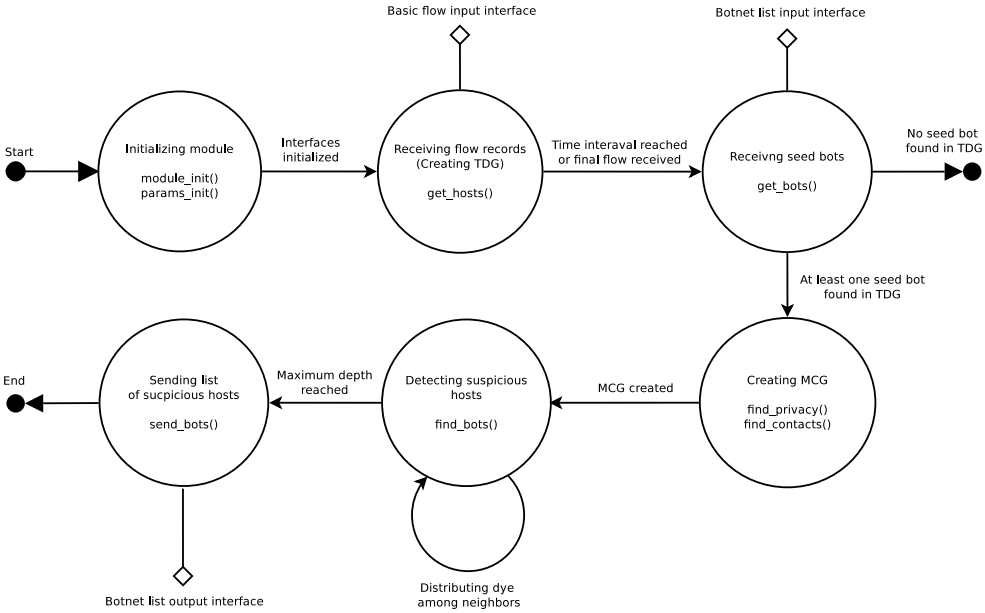

Figure 3.3: Graphical illustration of the module workflow

binary tree structure. If the host does not exist, the function creates the new host structure with new nodes in the binary tree based on the host's IP address. Afterwards the function creates an edge between source and destination host in the TDG indicating mutual communication. After a final record is received (data with zero size for testing purposes) or given time period is exceeded, the function returns a graph structure containing the created TDG.

4. Function *get_bots()* is called to receive bot list or list of suspicious hosts on the second input interface. The function seeks through the TDG if the bot is present in the current time period. All local hosts found on the bot list form the initial set of seed bots. In case of suspicious external host, all local peers of the given host are also suspicious, thus they are added to the initial set of seed bots. After the function returns the set of seed bots, the main function checks if the set is not empty.

5. Function *find_privacy()* is called to remove all popular external hosts from the TDG above given privacy-threshold as they might result in false positives. The function also removes all edges to the popular peer.

6. Function *find_contacts()* is called to create the MCG from the given TDG. The function connects all local hosts with an edge if both hosts shares at least one mutual contact. The edge is weighted by number of their mutual contacts.

7. Function *find_bots()* is called to perform the dye-pumping algorithm. The pseudo-code of the function is given in (1). The function returns the list of suspicious hosts above the given threshold.

8. Function *send_bots()* is called to send the list of suspicious hosts evaluated by the dye-pumping algorithm. The function sends the list on the output interface to be further examined. Another instance of the same module can be connected to the output interface, thus it might reveal the whole P2P botnet structure step-by-step.

9. Function *print_graph()* is called to log the information about the graphs based on a verbosity level. Brief level logs only the count of local, external and suspicious hosts. Basic level in addition logs all suspicious hosts (IP address and botnet probability) and local hosts (IP address, number of peers and number of mutual contacts). Advanced levels in addition add lists of peers (IP address or domain name if exists) and mutual contacts to each local host (IP address and number of mutual contacts with the given host). The function does not log the information by default as it might be very space-consuming for large amount of hosts.

Chapter 4

Testing and Evaluation

In this chapter, I will test and evaluate the realized module for P2P botnet detection. First, I will analyze the source code using dynamic analysis. The most important part appears in the section 4.3 where the module is demonstrated on the sample of real network traffic merged with synthetic P2P botnet communication.

4.1 Source Code Dynamic Analysis

Since the module is memory-consuming for very large amount of hosts, every memory allocation is checked for memory availability. Valgrind [29] has been used to perform dynamic analysis of the module. Valgrind is a framework for building dynamic analysis tools. All errors and memory leaks discovered by Valgrind have been fixed during the development. Valgrind has been executed with verbose parameter (`-v`) and full check for memory leaks (`--leak-check=full`). The example of Valgrind output is shown in Figure 4.1.

4.2 Dataset

For the testing and the evaluation of the module, only simulated data has been used. This data have been created by authors of the Malware Capture Facility Project [17]. The project focuses on capturing, analyzing and publishing real malware traffic. Datasets of the project have been captured on the Czech Technical University (CTU) network.

The module has been tested with annotated dataset containing botnet malware traffic. The dataset contains malicious P2P botnet communication among normal background traffic. All the flows are annotated whether it is a botnet or normal communication. The size of file with tested dataset is

```
==64160==
==64160== HEAP SUMMARY:
==64160==    in use at exit: 0 bytes in 0 blocks
==64160==  total heap usage: 6,099,832 allocs, 6,099,832 frees,
297,288,817 bytes allocated
==64160==
==64160== All heap blocks were freed -- no leaks are possible
==64160==
--64160-- used_suppression:      6 dl-hack3-cond-1
==64160==
==64160== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 6 from 6)
```

Figure 4.1: The output of Valgrind which has been used for the source code dynamic analysis. There are no memory leaks even though almost 300 MB have been allocated for the graph structures.

800 MB and it contains flow records in plain text. A sample of the dataset is given in Figure 4.2.

```
Date flow start          Durat  Prot   Src IP Addr:Port
2011-08-17 12:30:02.956  0.000  UDP    58.115.93.92:4056     ->
2011-08-17 12:30:02.956  0.000  TCP    147.32.84.165:21      ->
2011-08-17 12:30:02.958  0.085  TCP    87.98.230.229:80      ->
2011-08-17 12:30:02.958  0.008  TCP    147.32.86.195:40018   ->
```

```
Dst IP Addr:Port     Flags Tos Packets Bytes   Flows Label
147.32.86.165:12114  INT   0   1       60      1     Background
147.32.96.45:2049    RA_   0   1       60      1     Botnet
147.32.86.20:3465    FPA_  0   19      24441   1     Background
74.125.232.213:443   FPA_  0   3       225     1     LEGITIMATE
```

Figure 4.2: A sample of the dataset containing annotated flow records. The sample is split as the paper is not wide enough to print the whole line. The dataset can be downloaded at the site of the project, botnet dataset 50 [17].

The dataset consists of 8,087,513 flow records. During the 5 hours of observation, 115,415,230 packets have been transmitted forming a traffic in size of 98 GB. If the flows are measured on the perimeter of a network, there will be 334,887 local hosts and 344,977 external hosts. Unfortunately, this is a

misleading information because the direction flag is not provided within this dataset, thus the hosts appear on both sides.

The dataset has been created by merging of normal traffic with synthetic botnet topology. There are 10 bots in total which have been running on the virtual machines. After the initialization of the seed bot, it infects the other hosts with malware. The seed bot which infects other hosts has IP address 147.32.84.165. The other hosts have IP addresses within a range from 147.32.84.191 to 147.32.84.193 and from 147.32.84.204 to 147.32.84.209. All bots communicate with legitimate peers as well. The issue with this dataset is that it contains only a few bots in comparison to amount of hosts.

4.3 Evaluation

For the testing purposes of the module, the data has been modified into comma-separated values (CSV) using Unix shell script. After the modification, the data has been converted to a binary file using special modules from the Nemea framework.

In the given dataset environment, the proposed module achieved flawless results. All bots have been successfully identified without any false positives. However, only few bots have been participating in the communication. More datasets can be found in [17], but they also contain the same set of bots. Unfortunately, P2P botnet datasets are usually not publicly accessible. Hence, it can not be confirmed that the module successfully detects P2P botnet C&C communication with certain false positive rate. This issue is the main goal for the future work.

The module has been running 84.05 seconds in average. The delay is caused by massive amount of flow records in the dataset within a 5 hours time interval. The process of creating TDG is the most time consuming for thousands of hosts. The module is particularly designed for 5 minutes time windows. All parameters used bye dye-pumping algorithm have been set to default and the basic logging has been enabled (`-L 2`). The log output of the module is shown in Figure 4.3.

The bot with IP address 147.32.84.165 has been sent as the initial seed bot. From the output is clear that all the bots received some amount of dye from the seed bot which is higher than given minimum dye-threshold. At the bottom of the output, it is also shown that the seed bot shares 53 mutual contacts. The seed bot has communicated with 2408 peers during the time period. All the peers of local hosts have been filtered by the privacy-threshold, thus the local hosts might have communicated with more external hosts than the output shows. In this scenario, the choice of the seed bot is completely independent. The module provides the same result regardless the initial seed bot IP address.

```
Number of local hosts:                                  334887
Number of external hosts:                               344977
Number of local suspicious bots:                            10

Dye-pumping algorithm parameters:
* Privacy threshold:                                       128
* Breadth-first search depth:                                3
* Degree sensitivity coefficient:                          2.0
* Dye distribution factor:                                0.50
* Minimum dye threshold:                                  0.01
* Distributed dye among hosts:                            1.00

Known or suspicious local bots:
* Bot IP address:                                147.32.84.165
* Bot dye:                                            0.500000
* Bot IP address:                                147.32.84.208
* Bot dye:                                            0.069309
* Bot IP address:                                147.32.84.207
* Bot dye:                                            0.069201
* Bot IP address:                                147.32.84.206
* Bot dye:                                            0.065897
* Bot IP address:                                147.32.84.192
* Bot dye:                                            0.065870
* Bot IP address:                                147.32.84.191
* Bot dye:                                            0.061993
* Bot IP address:                                147.32.84.193
* Bot dye:                                            0.059805
* Bot IP address:                                147.32.84.204
* Bot dye:                                            0.039485
* Bot IP address:                                147.32.84.205
* Bot dye:                                            0.033132
* Bot IP address:                                147.32.84.209
* Bot dye:                                            0.030136

Local hosts:
* Local IP address:                              147.32.84.165
* Number of peers:                                         2408
* Number of mutual contacts:                                53
---------------------- output omitted ----------------------
```

Figure 4.3: Log output of the module.

Conclusion

The task of this bachelor's thesis has been given to study the botnet phenomenon, implement a module for P2P botnet detection and test the module on real traffic data. This text has described the network security issue of botnets and the process of module development for P2P botnet detection.

During the work on this project, I have studied and analyzed the botnet issue in computer networks. In this thesis, I have described and compared different botnet detection techniques proposed in the past. As the chosen approach, I have used the approach proposed in [13, 27] to create the generic module for detection of suspicious C&C botnet communication within a certain time window. The approach uses graph-based detection method by creating mutual contacts graph and performing dye-pumping algorithm which spreads the initial dye among suspicious neighbors.

The module is designed to be part of the Nemea framework which monitors CESNET2 network. For the testing of the module, I have used the real network data merged with synthetic P2P botnet topology [17]. The dataset has been captured on the CTU network within the malware capture facility project. Within this dataset, I have achieved excellent results. All the bots have been detected using only one seed bot as the input information.

Future work

In the future work, I will focus on the final deployment to the Nemea framework and testing the module in the real traffic environment. The module depends on the initial information of seed bots, thus it requires to communicate with another botnet detection module. Hence, I will try to implement another P2P botnet detection module using different technique to form a reliable detection of suspicious hosts together.

There are also many other opportunities to improve the current module. One of them is to lower memory resources as large amount of pointers are allocated in order to create graph structures. As the speed of the network

grows every year, more flows have to examined at the same time. Finding mutual contacts might be time consuming for higher privacy-threshold, thus the function can be parallelized. Another great opportunity might be using machine learning technique to set parameters in a automated way based on the previous results.

Bibliography

[1] Free Software Foundation, Inc: GNU Project - Free Software Foundation (FSF). 1983–2014. Available from: <http://www.gnu.org/software/>

[2] Verisign, Inc.: Verisign, Inc. 1995–2014. Available from: <http://www.verisigninc.com>

[3] CESNET, z.s.p.o: CESNET, z.s.p.o. 1996–2014. Available from: <http://www.cesnet.cz>

[4] Sourcefire, Inc.: Snort. 1998-2014. Available from: <http://www.snort.org>

[5] Napster, LLC: Napster. 1999–2011. Available from: <http://www.napster.com>

[6] Skype Technologies S.A.: Skype. 2003–2014. Available from: <http://www.skype.com>

[7] Satoshi Nakamoto: Bitcoin. 2009–2014. Available from: <http://www.bitcoin.org>

[8] Bartoš, V.; Žádník, M.; Čejka, T.: Nemea: Framework for stream-wise analysis of network traffic. Technical report, CESNET, a.l.e., 2013. Available from: <http://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf>

[9] Binkley, J. R.; Singh, S.: An algorithm for anomaly-based botnet detection. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, 2006, pp. 43–48.

[10] Chang, S.; Daniels, T. E.: P2P botnet detection using behavior clustering & statistical tests. In *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence*, ACM, 2009, pp. 23–30.

[11] Choi, H.; Lee, H.; Lee, H.; etc.: Botnet detection by monitoring group activities in DNS traffic. In *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, IEEE, 2007, pp. 715–720.

[12] Claise, B.: Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information. 2008.

[13] Coskun, B.; Dietrich, S.; Memon, N.: Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACM, 2010, pp. 131–140.

[14] Dagon, D.: Botnet detection and response. In *OARC workshop*, volume 2005, 2005.

[15] Dittrich, D.; Dietrich, S.: Command and control structures in malware. *Usenix magazine*, volume 32, no. 6, 2007.

[16] Feily, M.; Shahrestani, A.; Ramadass, S.: A survey of botnet and botnet detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*, IEEE, 2009, pp. 268–273.

[17] García, S.; Uhlíř, V.: Czech Technical University ATG Group: Malware Capture Facility Project. 2013–2014. Available from: <http://mcfp.weebly.com>

[18] Gu, G.; Perdisci, R.; Zhang, J.; etc.: BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In *USENIX Security Symposium*, 2008, pp. 139–154.

[19] Gu, G.; Zhang, J.; Lee, W.: BotSniffer: Detecting botnet command and control channels in network traffic. 2008.

[20] Karasaridis, A.; Rexroad, B.; Hoeflin, D.: Wide-scale botnet detection and characterization. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, volume 7, Cambridge, MA, 2007.

[21] Kristoff, J.: Botnets. In *32nd Meeting of the North American Network Operators Group*, 2004.

[22] Liao, W.-H.; Chang, C.-C.: Peer to peer botnet detection using data mining scheme. In *Internet Technology and Applications, 2010 International Conference on*, IEEE, 2010, pp. 1–4.

[23] Masud, M. M.; Al-Khateeb, T.; Khan, L.; etc.: Flow-based identification of botnet traffic by mining multiple log files. In *Distributed Framework and Applications, 2008. DFmA 2008. First International Conference on*, IEEE, 2008, pp. 200–206.

[24] Nagaraja, S.; Mittal, P.; Hong, C.-Y.; etc.: BotGrep: Finding P2P Bots with Structured Graph Analysis. In *USENIX Security Symposium*, 2010, pp. 95–110.

[25] Provos, N.: A Virtual Honeypot Framework. In *USENIX Security Symposium*, volume 173, 2004.

[26] Ramachandran, A.; Feamster, N.; Dagon, D.: Revealing botnet membership using DNSBL counter-intelligence. *Proc. 2nd USENIX Steps to Reducing Unwanted Traffic on the Internet*, 2006: pp. 49–54.

[27] Ruehrup, S.; Urbano, P.; Berger, A.; etc.: Botnet detection revisited: Theory and practice of finding malicious P2P networks via Internet connection graphs. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, IEEE, 2013, pp. 435–440.

[28] Schonewille, A.; van Helmond, D.-J.: The domain name service as an IDS. *Research Project for the Master System-and Network Engineering at the University of Amsterdam*, 2006.

[29] Seward, J.; Nethercote, N.; Fitzhardinge, J.: Valgrind, an open-source memory debugger for x86-GNU/Linux. *URL: http://www.ukuug.org/events/linux2002/papers/html/valgrind*, 2004.

[30] Steggink, M.; Idziejczak, I.: Detection of peer-to-peer botnets. *University of Amsterdam, February*, 2008.

[31] Stone-Gross, B.; Cova, M.; Gilbert, B.; etc.: Analysis of a botnet takeover. *Security & Privacy, IEEE*, volume 9, no. 1, 2011: pp. 64–72.

[32] Strayer, W. T.; Lapsely, D.; Walsh, R.; etc.: Botnet detection based on network behavior. In *Botnet Detection*, Springer, 2008, pp. 1–24.

[33] Zhao, D.; Traore, I.; Ghorbani, A.; etc.: Peer to Peer Botnet Detection Based on Flow Intervals. In *Information Security and Privacy Research*, Springer, 2012, pp. 87–102.

Appendix A

Acronyms

API Application Programming Interface

BFS Breadth-First Search

C&C Command and Control

CLI Command-Line Interface

CPU Central Processing Unit

CTU Czech Technical University

DDoS Distributed Denial of Service

DNS Domain Name System

DNSBL DNS-based Black-hole List

FTP File Transfer Protocol

GUI Graphical User Interface

IDS Intrusion Detection System

IPFIX IP Flow Information Export

IRC Internet Relay Chat

ISP Internet Service Provider

MCG Mutual Contacts Graph

MiM Man in the Middle

Nemea Network Measurement Analysis

NREN National Research and Education Network

P2P Peer-to-peer

TCP Transport Control Protocol

TDG Traffic Dispersion Graph

TRAP TRaffic Analysis Platform

UDP User Datagram Protocol

UniRec Unified Record

Appendix B

Installation Manual

This installation manual has two sections. In the first section, there are described dependencies of the module. In the second section, Unix commands are provided to successfully install the module.

B.1 Dependencies

The module depends on some Nemea libraries mentioned in section 2.3. In order to compile the module for P2P botnet detection, some Nemea libraries have to be installed. Current version of the Nemea framework can be downloaded at `https://www.liberouter.org/technologies/nemea/`. This is the list of the module dependencies:

- libtrap
- libunirec
- libm

B.2 Module installation

The module has been created using the GNU build system also known as Autotools [1]. It is a suite of tools for build automation. The Autotools system makes the C programs portable among Unix-like systems. It is simple to use and configure. To successfully install the module, unpack the source codes located in */src* directory on the CD. Afterwards, in the source directory of the module type following commands:

```
./configure
make
make install
```

Appendix C

Contents of Enclosed CD

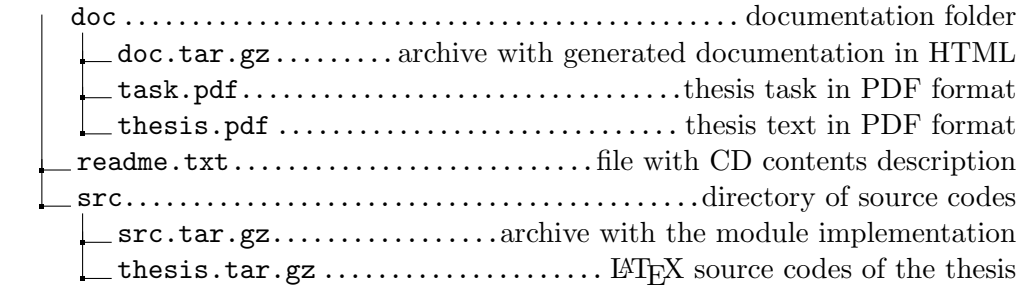

Figure C.1: Contents of enclosed CD