



Norwegian University of  
Science and Technology

# Visualizing Spatial and Temporal Dynamics of a Class of IRC-Based Botnets

**Andreas Hegna**

Master of Science in Communication Technology

Submission date: June 2010

Supervisor: Svein Johan Knapskog, ITEM

Co-supervisor: Stein Nilsen, FFI



# Problem Description

The amount of useful information that can be extracted from a visualization, depends on the underlying metrics. Developing an understanding of the nature of the data is an essential part of constructing new visualization methods. The first part of the study must necessarily contain a description and analysis of botnets from the viewpoint of constructing visualization metrics. The second part will focus on the visualization of botnet dynamics, including a short survey of network visualization tools and methods. Depending on the availability of data and/or simulation resources it will be up to the student whether to focus on the analysis of real data, or developing and illustrating concepts using simulated data.

Assignment given: 25. January 2010  
Supervisor: Svein Johan Knapkog, ITEM



# Abstract

Botnets are a serious threat to the security of personal computers, businesses and even countries. They can launch attacks on remote systems and infrastructure, perform espionage and once installed they essentially hand over control of the computer to a botnet administrator. It is very difficult to detect their presence in a network as it is hard to distinguish their footprint from normal traffic. By using Internet Relay Chat (IRC)-based botnets as an example for visualizing spatial and temporal dynamics, I will attempt to detect the presence of a bot and visualize the results. Based on previous works for detecting malware, the choice was made to use *process-to-port* mapping as the base metric for visualization. Investigation into botnets was an integral part of the thesis. Published sources along with research into botnet administrator communities were used to provide a solid information base. A bot application, which is part of a botnet, can be regarded in the same way as any other piece of software, but with added functionality for communication and remote control. As such, it is bound by the same proprietary technologies. The thesis focuses on a method of detection that relies on IP and port pairs with host computer metrics, which can be expanded to a distributed context with the use of Simple Network Management Protocol (SNMP). The software for *process-to-port* mapping and visualization of a botnet has been developed with focus on geographical location. Use of geography for the Visualization application proved to be a good choice and communicating applications are shown in a distinct and clear way. Experiments conducted, successfully detected and visualized the bot communicating with the command and control server as metrics were collected on a host machine. The developed visualization software also shows general network activity and has potential to be used in a more general context. It is concluded that given some preconditions with regards to a bot's rootkit capabilities, detection of a

botnet is successful. Given the availability of certain SNMP OIDs, it is possible to perform botnet detection and general network visualization in a large scale and distributed context.

# Preface

I started by focusing on how to detect and visualize botnets, the idea was to focus on detection and implement some ideas for detection based on machine usage profiles and visualize the results in a way that was easy to understand. I looked into using Snort logs as a part of the detection scheme, but as the thesis evolved I focused less on finding new ways of detection as there were a number of papers on this subject already. Instead the focus shifted to visualization and *process-to-port* mapping as I believe that I could contribute with new material in this area. The original though was to create something that could be used in a distributed context and that could be run with multiple computers. To limit the amount of work and still keep a distributed context in mind, the choice was made to use technologies designed to be distributed but develop the visualization to function on one host computer.

A great deal of time was spent investigating communities where botnets were sold and traded. Botnets from these forums are included in the thesis to keep the material as relevant as possible. Some of the botnet authors own descriptions of how bots work are also included to give some insight into technology from their perspective.

In total I wrote about 8500 lines of code for the *process-to-port* mapping and about 2500 lines of code for the visualization software. It turned out to require significantly more work than anticipated to cover both topics and I ended up giving each topic the same amount of focus. Being given the opportunity to present my work for the North Atlantic Treaty Organization (NATO) AC/323 (RTO) IST085/RTG041 group on “Interactive Visualisation of Network Dynamics” was a great motivator to try and create something new. Based on the feedback that I got from the NATO group I believe I succeeded in doing this. I hope this

thesis and its findings will benefit others who want to use host based metrics and visualization for botnets and network traffic.

## Acknowledgements

I would like to thank my professor, Svein Johan Knapskog, for providing good advice and feedback as well as allowing me to use the Norwegian University of Science and Technology (NTNU) Quantifiable Quality of Service (Q2S) security group for input on the work.

I would also like to thank my supervisor, Dr. Stein Nilsen (Principal Scientist for the Norwegian Defence Research Establishment (FFI)), for providing good advice and feedback on the work as well as advice on writing a great thesis. The opportunity given by him to present my thesis at the NATO meeting on “Interactive Visualisation of Network Dynamics” was a real highpoint the semester and is something that I will cherish.

I would also like to point out the great support in the Carnivore Forums by the authors of Carnivore and I especially want to thank Alexander Galloway, (galloway@nyu.edu), at New York University for providing support and the source code for Carnivore library. This helped me to better integrate it with my visualization.



# Contents

<b>Abstract</b>	<b>III</b>
<b>Preface</b>	<b>V</b>
<b>Table of Contents</b>	<b>V</b>
<b>List of Figures</b>	<b>IX</b>
<b>List of Tables</b>	<b>XIII</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Limitations . . . . .	2
1.2 Methodology . . . . .	2
1.3 Structure . . . . .	4
<b>2 Botnet</b>	<b>7</b>
2.1 Introduction . . . . .	7
2.2 Botnet evolution . . . . .	9
2.3 Detection . . . . .	10

2.4	Java based Botnet and the theory of the Fully UnDetectable (FUD) botnet . . . . .	12
2.5	Command and Control (C2) Structure and Topology . . . . .	12
2.5.1	Centralized topologies . . . . .	15
2.5.2	Decentralized topologies . . . . .	21
2.6	Botnet metrics . . . . .	23
2.6.1	Terminology . . . . .	23
2.6.2	TCP/IP Model . . . . .	24
2.6.3	Packet Capture and Analysis . . . . .	31
2.6.4	Source/Destination Confusion . . . . .	37
2.6.5	Operating System (OS) metrics . . . . .	38
2.7	Simple analysis of a Java Based bot . . . . .	45
2.7.1	Botnet setup . . . . .	46
2.7.2	Botnet startup . . . . .	49
2.7.3	Botnet commands . . . . .	50
2.7.4	Summary for bot analysis . . . . .	52
<b>3</b>	<b>Visualization</b>	<b>55</b>
3.1	Introduction . . . . .	55
3.2	Visualization metrics . . . . .	57
3.2.1	Geolocation . . . . .	58
3.3	Technology . . . . .	61
3.3.1	Processing . . . . .	61
<b>4</b>	<b>Experiment</b>	<b>67</b>
4.1	Introduction . . . . .	67

4.2	Materials and experiment setup . . . . .	68
4.2.1	Equipment . . . . .	68
4.2.2	Experiment Setup . . . . .	69
4.2.3	Setup for IRC server and channel . . . . .	69
4.2.4	Modification of the “Family Photos” bot . . . . .	72
4.2.5	Setup for PC1, a bot infected computer . . . . .	74
4.2.6	Setup for PC2, a bot administrator computer . . . . .	75
4.3	Methods . . . . .	76
4.3.1	Step by step guide for PC2 . . . . .	76
4.3.2	Step by step guide for PC1 . . . . .	78
4.4	Results . . . . .	79
4.4.1	Results for PC1 . . . . .	79
4.4.2	Results for PC2 . . . . .	84
4.5	Analysis . . . . .	88
4.5.1	Analysis PC1 . . . . .	88
4.5.2	Analysis PC2 . . . . .	90
4.5.3	Summary analysis . . . . .	91
<b>5</b>	<b>Discussion</b>	<b>93</b>
<b>6</b>	<b>Conclusion</b>	<b>99</b>
<b>7</b>	<b>Future work</b>	<b>101</b>
	<b>Literature</b>	<b>103</b>

<b>A</b>	<b>Definitions</b>	<b>113</b>
<b>B</b>	<b>Software</b>	<b>115</b>
B.1	Starting point . . . . .	115
B.2	Process2Port . . . . .	117
B.2.1	process2port.objects package . . . . .	118
B.2.2	process2port package . . . . .	118
B.2.3	cmdexecute package . . . . .	119
B.2.4	services package . . . . .	120
B.2.5	snmp package . . . . .	121
B.3	Visualization . . . . .	121
B.3.1	Key commands and functionality . . . . .	122
B.3.2	Visualization functionality . . . . .	128
B.3.3	Example usage . . . . .	137
<b>C</b>	<b>SNMP installation</b>	<b>140</b>
C.1	Install SNMP Server on Windows 7 . . . . .	140
C.2	Install Getif on Windows 7 . . . . .	141
<b>D</b>	<b>Excerpt botnet source code</b>	<b>143</b>
D.1	a.class . . . . .	143
D.2	main.class . . . . .	145

# List of Figures

2.1	Computer sales per day vs. infections per day in 2009. . . . .	8
2.2	A botnet author's own description of what is perceived to happen under normal execution of exe files. . . . .	13
2.3	A botnet author's own description of what is perceived to happen under normal execution of Java ARchive (JAR) files. . . . .	14
2.4	Star C2 topology with direct communication between the central command server and each bot agent. [67, p. 2] . . . . .	16
2.5	Multi-server C2 topology with direct communications between each distributed cluster of command servers and each bot agent. [67, p. 3] . . . . .	19
2.6	Hierarchical C2 topology with proxied C2 communication between the botnet administrator and bots. [67, p. 4] . . . . .	20
2.7	Random C2 topology with no centralized C2 server infrastructure. [67, p. 5] . . . . .	22
2.8	The TCP/IP model's four layers of abstraction show with typical protocols used in each layer. . . . .	25
2.9	Differences between the Open System Interconnection (OSI) and TCP/IP model. . . . .	26
2.10	Internet Protocol (IP) datagram that shows the data is repackaged with the lower layers header. . . . .	27

2.11	IP Datagram with IPv4 header and data field. [70, p. 11]	29
2.12	Transmission Control Protocol (TCP) Datagram with TCP header and data field. [71, p. 15]	31
2.13	Example of routers that forward, but do not process Transport Layer data.	32
2.14	Simple graph showing the concept of source/destination confusion.	37
2.15	Excerpt of the available botnet commands for the “Family Photos” bot.	47
2.16	VirusTotal scan results for <i>jusched.jar</i>	53
3.1	A world map projection from the NASA Blue marble project. [60]	59
3.2	The rendered 3D representation of the Earth with Processing.	62
3.3	Only 129 lines of code can generate a high quality 3D representation of earth in processing.	64
3.4	The Carnivore Library for Processing, displaying a packet dump. [23]	65
4.1	Basic setup for the experiment.	70
4.2	VirusTotal scan results for <i>ModifiedJavaBot.jar</i> .	73
4.3	Extended experiment setup.	77
4.4	ipconfig results for PC1.	80
4.5	Initial Taskmanager results for PC1.	80
4.6	Windows Security Alert for Java Platform SE binary.	81
4.7	netstat results for PC1 before and after <i>ModifiedJavaBot.jar</i> has been initiated.	82
4.8	Taskmanager results for PC1 after <i>ModifiedJavaBot.jar</i> has been initiated.	82
4.9	Visualization showing IRC communication between <i>ModifiedJavaBot.jar</i> and IRC server in South Africa.	83

4.10	Visualization zoomed in on IRC server in South Africa. . . . .	84
4.11	Visualization of connections made by <i>ModifiedJavaBot.jar</i> when PC2 issues the <i>.getip</i> command. . . . .	85
4.12	Windump results for PC1. . . . .	86
4.13	ipconfig results PC2. . . . .	86
4.14	Windows Security Alert for mIRC. . . . .	87
4.15	IRC chat conversation between PC1 and PC2 with issued commands. . . . .	87
4.16	Windump results for PC2. . . . .	88
4.17	PC1 Wireshark request with the <i>.getip</i> command. . . . .	89
4.18	PC1 Wireshark response to the <i>.getip</i> command. . . . .	89
4.19	PC2 Wireshark request with the <i>.getip</i> command. . . . .	90
4.20	PC2 Wireshark response to the <i>.getip</i> command. . . . .	90
B.1	Original visualization provided by Carnivore library. [23] . . . . .	116
B.2	Process2Port package diagram. . . . .	117
B.3	Visualization after roughly 4 minutes running time. Running Peer-to-Peer (P2P) applications at time of visualization include Skype, Spotify and uTorrent. . . . .	123
B.4	Zooming and panning the map. Here showing an unknown connection. . . . .	125
B.5	Carnivore console log showing IP and port pairs as well as TCP Flags. . . . .	126
B.6	Visualization showing a list of mapped applications. . . . .	127
B.7	Visualization of a “raw” packet capture. . . . .	129
B.8	Visualization with host IP filtering resulting in reduced packets. . . . .	130
B.9	Visualization showing mapping of image icon to port number. . . . .	132
B.10	Visualization showing process to port mapping. . . . .	134

B.11 Visualization showing unmapped applications and resulting in “Missing icon for image name” label. . . . .	135
B.12 Visualization showing list of mapped applications in upper left corner. . . . .	136
B.13 Example visualization of captured network traffic with mapped mIRC connection. . . . .	139



# List of Tables

B.1	Visualization key commands . . . . .	124
B.2	Setup key commands . . . . .	126
B.3	Various commands . . . . .	127
B.4	Rules commands . . . . .	127



# Acronyms

<b>API</b>	Application Programming Interface
<b>ARP</b>	Address Resolution Protocol
<b>C2</b>	Command and Control
<b>CSV</b>	Comma-separated values
<b>DDoS</b>	Distributed Denial of Service
<b>DNS</b>	Domain Name System
<b>DOD</b>	Department of Defense
<b>FTP</b>	File Transfer Protocol
<b>FUD</b>	Fully UnDetectable
<b>GT</b>	Global Threat
<b>HTTP</b>	HyperText Transfer Protocol
<b>ICMP</b>	Internet Control Message Protocol
<b>IDC</b>	International Data Corporation
<b>IDPS</b>	Intrusion detection and prevention systems
<b>IDS</b>	Intrusion Detection System
<b>IP</b>	Internet Protocol
<b>IPS</b>	Intrusion Prevention System

**IRC** Internet Relay Chat  
**ISN** Initial Sequence Number  
**IRTF** Internet Research Task Force  
**ISO** International Organization for Standardization  
**ISP** Internet Service Provider  
**IT** Information Technology  
**JAR** Java ARchive  
**JVM** Java Virtual Machine  
**LAN** Local Area Network  
**MAC** Media Access Control  
**Malware** Malicious Software  
**MIB** Management information base  
**NATO** North Atlantic Treaty Organization  
**NTNU** Norwegian University of Science and Technology  
**OID** object identifier  
**OS** Operating System  
**OSI** Open System Interconnection  
**P2P** Peer-to-Peer  
**PID** Process IDentifier  
**Q2S** Quantifiable Quality of Service  
**RARP** Reverse Address Resolution Protocol  
**REGEX** REGular EXpressions  
**RFC** Request for Comments  
**SDK** Software Development Kit

**SNMP** Simple Network Management Protocol  
**TCP** Transmission Control Protocol  
**TTL** Time-To-Live  
**UDP** User Datagram Protocol  
**URL** Uniform Resource Locator  
**UTM** Universal Transverse Mercator  
**WMI** Windows Management Instrumentation  
**WMIC** Windows Management Instrumentation Command-line  
**XML** Extensible Markup Language



# Chapter 1

## Introduction

“Throughout the centuries there were men who took first steps, down new roads, armed with nothing but their own vision.”—Ayn Rand

Botnets are one of the biggest threats one can face on the Internet and thus much work has gone into detecting these networks. The different approaches will be discussed in more detail in Section 2.3, but among these approaches there are some that stand out with regards to this thesis. “A Host-Based Approach to BotNet Investigation?” a practical, although manual, approach for botnet detection for digital investigators is presented. [36] With analysis of packet captures as well as monitoring running processes, botnets were successfully detected. I believe that this approach holds merit and by taking the principles from that paper, it would be possible to contribute to this area of botnet detection. In Chapter 2, botnets will be explored in more detail.

Representing data is also a great challenge because the amount of information available can be vast. Visualization provides a way for representing data in different ways and thus might yield added value to the user with regards to finding new points of interest or abstracting away information. One note about the naming of the visualization software is that when the text references “Visualization”, with a capital letter, it refers to the developed application. In Chapter 3, visualization will be explored in more detail.

## 1.1 Limitations

To be able to limit the scope of the problem description and narrow the focus on the thesis, further limitations are needed. I will attempt to automatically do *process-to-port* mapping with technologies that support a distributed context as a way of detecting botnets, but to limit the scope these technologies will be implemented to function on one host machine.

Raw packet capture that is manipulated by host based metrics will be the main focus of the thesis. In addition, visualization of the metrics will be performed in order to give the user the possibility to discover patterns in the visualization. How to best detect the characteristics of botnets will be discussed and based on the discussed information, selected tools and methods will be used to form the base metrics for detection and visualization. Ideas from “A Host-Based Approach to BotNet Investigation?” [36] and the principle of *locality* by Robert Lee and Sheau-Dong Lang [37] will also be explored.

The major focus of the thesis will be how to visualize spatial and temporal dynamics in a way that is stimulating and informative to the user. By spatial dynamics the information will be displayed according to its relation to the host computer. By temporal dynamics the information will be showed based on what time the information was captured. Further limitations will be with regards to what type of botnet that will be used when attempting visualization. I will use an IRC-based botnet as they are among the best studied and are still very predominant in the wild.

## 1.2 Methodology

Firstly, information will be gathered from books, articles and papers. Forums where botnet authors communicate, botnets are sold and traded will be part of the thesis. Resources i collected from the botnet authors themselves will also give some insight into this technology from their perspective. A shot survey of visualization software available will also be presented. When an introduction into the background material is given, an analysis of available metrics will be explored before the experimentation phase will begin. Before experimenting with an actual botnet, analysis and visualization of network traffic will be carried out. The experiment with an actual botnet will be carried out in a controlled manner



and data will be collected for later analysis. The collected metrics will be input for the visualization and displayed in real time. Results from the experiment and the corresponding visualization will be discussed in the later chapters.

By using *process-to-port* mapping I expect to see each connection the host computers applications are responsible for. The botnet is expected to register with the computers system monitoring tools in the same way as any other application, given that a rootkit is not employed. If distinct and detectable changes are made to the system when the bot is initiated, these statements will be considered to be proven. These changes will consist of new entries in the system monitoring tools, log entries from packet captures as well as log entries and visuals from developed software. The resulting visualization method will display connections based on distinct applications and not based on traffic volume. Visuals are expected to produce distinct observational changes supported by the previous statements. If no changes are observed, these statements will be considered invalid.

Throughout the thesis I will utilize extensive citations in order to enable readers and those interested in the same topic, to get access to the same information and to be able to find the resources I describe.

In the attempt to visualize IRC-based botnets, I will follow as closely as possible the six steps of the information visualization process; *Define the problem*, *Assess available data*, *Process information*, *Visual transformation*, *View transformation* and *Interpret and decide*. [41, p. 120-121]

1. *Define the Problem*: The need to understand the problem and background on the topic is a vital part. To be able to find an answer, one must first know the questions surrounding the topic.
2. *Assess available data*: Next comes the process of finding out what data is available.
3. *Process information*: Next, one has to extract the relevant information found in the previous step.
4. *Visual transformation*: How should the visuals be presented?
5. *View transformation*: The visualization from the previous step is refined to focus on the most important aspects.

6. *Interpret and decide*: The last step is to look back on what has been generated and see if the results solve the initial problem.

## 1.3 Structure

The thesis is organized in the following structure:

- Chapter 1: Introduction

An introduction to the problem is presented and limitations are defined. The methodology used is explained in detail before an outline of the structure of the thesis is presented.

- Chapter 2: Botnet

Visualization cannot exist without data and knowledge about botnets. This chapter provides information and technologies employed by botnets. Metrics that are useful for detection and visualization are discussed, in addition to a short analysis of a real world bot.

- Chapter 3: Visualization

How the data is visualized is at the core of this thesis. This chapter discusses important aspects and metrics related to visualization. Ranging from how people understand the world around them, to relevant technologies for presenting the data.

- Chapter 4: Experiment

In order to address the initial problem of the thesis an experiment is carried out. The developed software is used in conjunction with a modified version of a real world bot. Results from this experiment forms the basis for the discussion and conclusion.

- Chapter 5: Discussion

This chapter discusses the findings from the experiment with relation to the previously presented material.

- Chapter 6: Conclusion

The findings from the thesis are presented in the final conclusion.

- Chapter 7: Future work

Possible future work on the findings presented in the thesis will be discussed in this chapter.

- Appendix A: Definitions

Various important definitions for understanding terms and technologies mentioned in this thesis. If the reader is not familiar with terms presented, one should reference this appendix.

- Appendix B: Software

A significant portion of the work went into creating the software used in the experiment. Details about the developed library for *process-to-port* mapping and the visualization can be found in this appendix.

- Appendix C: SNMP installation

A description of how to install and set up the SNMP agent on Windows 7 is described in this appendix.

- Appendix D: Excerpt botnet source code

The compiled byte code, from the analyzed bot from Chapter 2, is shown in this appendix.



# Chapter 2

## Botnet

“15 Feb 1943

If only it were possible to reproduce yourself a million times over so that you can achieve a million times more than you can today.”

–Dr. Paul Joseph Goebbels, Propaganda Minister for Nazi Germany

–From 15 Feb 1943 entry in the personal diary of Paul Joseph Goebbels.

### 2.1 Introduction

Botnets are one of the biggest threats one can encounter in cyberspace today. They can be, and often are, used as a weapon in the same way as conventional ordinance. One bot can be bad enough, but usually the network of bots is significantly bigger. As the quote at the start of this chapter tries to convey; one evil can be bad, but if you are faced with an enemy that has 1 million copies of itself, one will be dealing with an entirely different monster.

Botnets are emerging as the prominent threat to the security of companies and they reportedly threaten national infrastructure and security. [34] [44] More than 150,000 computers were reportedly infected daily during the second quarter of 2009. [45] A botnet consisting of one million bots, with the global average

connection speed of 1.7 Mbps [1, p. 12], can wield a crippling 1,7 terabits of traffic. This is enough to take down most Fortune 500 companies and even countries as shown in the 2007 cyber attack on Estonia, where the Estonian parliament, ministries, banks and media were targeted. [18]

According to an International Data Corporation (IDC) [31] press release from 2009; 284.1 million computers were forecast to be sold in 2009. If all the infections per day would be from the 778 365 computers sold daily, 150 000 of the newly purchased computers that day would be infected. This would mean 19% of the sold computers could potentially be infected as seen in Figure 2.1

### Total sales per day in 2009: 778,356

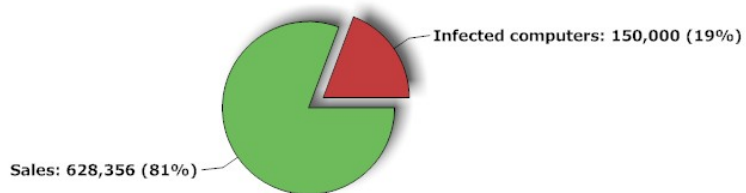


Figure 2.1: Computer sales per day vs. infections per day in 2009.

“Overall, botnets control more compromised machines than had been previously believed. Only a handful of criminals (likely a few hundred) have more than 100 million computers under their control. This means that cybercriminals have more computing power at their disposal than the entire world’s supercomputers combined. It’s no wonder then that more than 90 percent of all e-mail worldwide is now spam.”—Trend Micro [48]

To be able to have a common understanding of the challenges that this thesis presents, it is important to give some background knowledge of this Malicious Software (Malware).

There are several approaches to detecting botnets and compromised computers infected by them. To summarize there are three main ways of detecting botnets; *Signature- and Rule-based Detection*, *Behavior and Activity Detection* and *Command and Control (C2) Session Detection*.

## 2.2 Botnet evolution

The first worm that utilized IRC as a means of remote control was the *PrettyPark.Worm* in June of 1999, according to “The Evolution of Malicious IRC Bots” a Symantec white paper. It allowed the attacker to retrieve a variety of information about the infected system as well as the ability to download and execute files. [9, p. 6] [15, p. 7-8]

In late 2000 a new kind of bot named *Global Threat (GT) bot* appeared which used a hacked copy of the mIRC [39] client executable in combination with custom scripts to connect to a remote server and wait for commands. mIRC [39] is an IRC based software package that supports scripts and raw TCP and User Datagram Protocol (UDP) socket connections. One type of GT bot, Backdoor.IRC.Aladinz also had some interesting functionalities. It attempted to delete *netstat.exe* file from the Windows directory in order to hide its presence on the infected system. [9, p. 6-7] [15, p. 8-9]

In early 2002 *Sdbot* appeared with its own IRC client incorporated within its executable. It was written entirely in C++, could load itself via the Windows registry and using legitimate-looking process names, it could hide itself to a degree. [9, p. 9-13] [15, p. 9-10]

*Agobot*, also known as *Gaobot*, surfaced in 2002 and even if it was a different project than *Sdbot*, it had the majority of the same functionality however the bot was object oriented and it was more robustly developed. Later versions also utilized exploits for propagating itself, techniques for hiding itself using rootkit technology, encryption and polymorphism. *Agobot* uses IRC for Command and Control (C2), but it is spread through P2P file-sharing networks. Variations of *Agobot* include: *Phatbot* [83], *Forbot*, *Polybot*, and *XtremBot*. The source code also showed that functionality was divided into separate source files, it was distributed with the Microsoft Visual C++ project files and it included details on how one could obtain a made-to-order *Agobot* release. [9, p. 13-18] [15, p. 10-11] Selling of custom bots and botnets are frequently done in plain sight

according to my research into these communities.

*Spybot*, also known as *Milkit*, is built on *Sdbot* and emerged in 2003. It added spyware capabilities such as collecting logs of activity, data from web forms, key stroke logging, logging of everything copied to the Windows clipboard, killing antivirus processes and other security products, control of web cameras and included various exploits. [9, p. 20-30] [15, p. 12-14]

## 2.3 Detection

**Signature- and Rule-based Detection** is one of the most widely used choices. *Signature-based* detection schemes are based on recognizing something that has been seen and identified previously. The open source network intrusion prevention and detection system (IDS/IPS) called Snort [81] is one such solution. It is configured with a set of rules or signatures from log traffic that has been deemed suspicious. To be able to improve signature quality it is also useful to be able to *reverse-engineer* the actual bot source code, but it is very hard to get a hold of the entire program.

**Behavior and Activity Detection** is one of the more active areas with regards to proposing new solutions for detection. It tries to detect the presence of botnets by looking at their characteristics, how they interact with their environment or what the normal behavior of the system should be and see whether the activity is “un-normal” or “normal”. Some have proposed to identify the *botnet behavior* by identifying bot commands through Run-time execution monitoring. This could involve monitoring access to core windows *dll* files and other system variables. [68] Other researchers point to network behavior as a way of detecting botnets. [85] Claudio Mazzariello and Carlo Sansone proposed ways of detecting IRC botnets by means of describing behavior for specific classes of network users. This could involve monitoring the average number of vowels in a message and deduce what classifies as “human” chat participants. [43] Earlier work by Binkley and Singh proposed an anomaly-based algorithm for detecting IRC-based botnet meshes. [6] Another interpretation of establishing “normal” behavior for a system is based on the principle of *locality* which was proposed by Robert Lee and Sheau-Dong Lang. [37]

In the publication “Exploiting Temporal Persistence to Detect Covert Botnet Channels”, the authors proposed the use of a metric called *persistence* to capture



“lightweight” yet “regular” communication and by using a notion of “destination atoms”, in which multiple IP destinations are aggregated into a single domain name. According to the paper they were successful in detecting C2 traffic and this method also provide a possible solution for detecting “fast-flux” botnets. [24]

**Command and Control (C2) Session Detection** will attempt to analyze the packet payload, looking for C2 traffic or more specific; botnet command strings. Christopher Hanna [26] analyzed IRC packet payloads and could detect well known bots, but not variants that used different command strings. In the publication “A Host-Based Approach to BotNet Investigation?”, a practical way for digital forensics investigators to determine the presence of botnets by looking at host-based metrics, such as running processes that generate network traffic at startup. [36]

One problem with “A Host-Based Approach to BotNet Investigation?” [36] is that you would already need to have a suspicion about the specific computer and one computer at a time would be examined. The idea of running *process-to-port* mapping is not new, but this paper proves that it is possible to determine the presence of a botnet based on host-based metrics.

By looking at host-based metrics and excluding payload analysis for determining the presence of C2 traffic; I believe it can be used in a large scale network context where it is possible for an organization to impose a mandatory basic setup for data collection. By using visualization to reduce the workload and help with detection, it could be used as a supplement to established detection schemes as well as visualization. To the best of my knowledge, there has been no attempt to visualize specifically botnets. This paper will attempt to do just that and focus on extracting metrics. Based on information and lessons learned from “A Host-Based Approach to BotNet Investigation?” [36] as well as the importance of locality mentioned by Lee and Lang [37], I will attempt to detect botnets on a host machine and visualize the results. By visualizing the communication geographically, the visualization itself will hopefully prove more valuable.

## 2.4 Java based Botnet and the theory of the Fully UnDetectable (FUD) botnet

Although there have been numerous new botnets since 2003, the majority of botnets have been based on these early pioneers. If we jump forward to the present day, some of the developments in the botnet community are the emergence of Java based botnets. Due to the volatility of the sources, not all claims are cited.

The main theory behind the rapid growth of Java botnets seems to be the theory that Java is Fully UnDetectable (FUD) from anti-virus software. The arguments behind this statement is that where an *exe* file calls the Windows System Application Programming Interface (API) directly, the anti-virus software is monitoring who is requesting access and thus will be detected. An example description by a botnet administrator can be seen in Figure 2.2. With a Java ARchive (JAR) file however, the JAR file calls the Java Virtual Machine (JVM), which in turn calls the Windows System API. According to some botnet administrators it is not possible to use a signature search on JVM and therefore they are unaffected by anti-virus software, as described in Figure 2.3.

There are a few Java based botnets in the wild, but it does not seem to be known who made them as botnet authors often bicker over who made what. Some tutorials on how to use and administer botnets have been published, and one such botnet, that will be called “*family photos*” based on the archive filename it was shared under, will be used in the experiments conducted in this thesis. One Java based bot has also been analyzed by Craig Williams at Cisco [102]. This code is similar to “family photos”, but is presumably the “IRC” bot described in Section 2.7, due to source code similarities.

## 2.5 Command and Control (C2) Structure and Topology

Botnets come in many different implementations and use a variety of different C2 topologies that each has its own strengths and weaknesses. The chosen topology is typically a choice between the perceived risk by the botnet administrator, the ease-of-use and the economic model of the particular botnet. [67, p. 1]

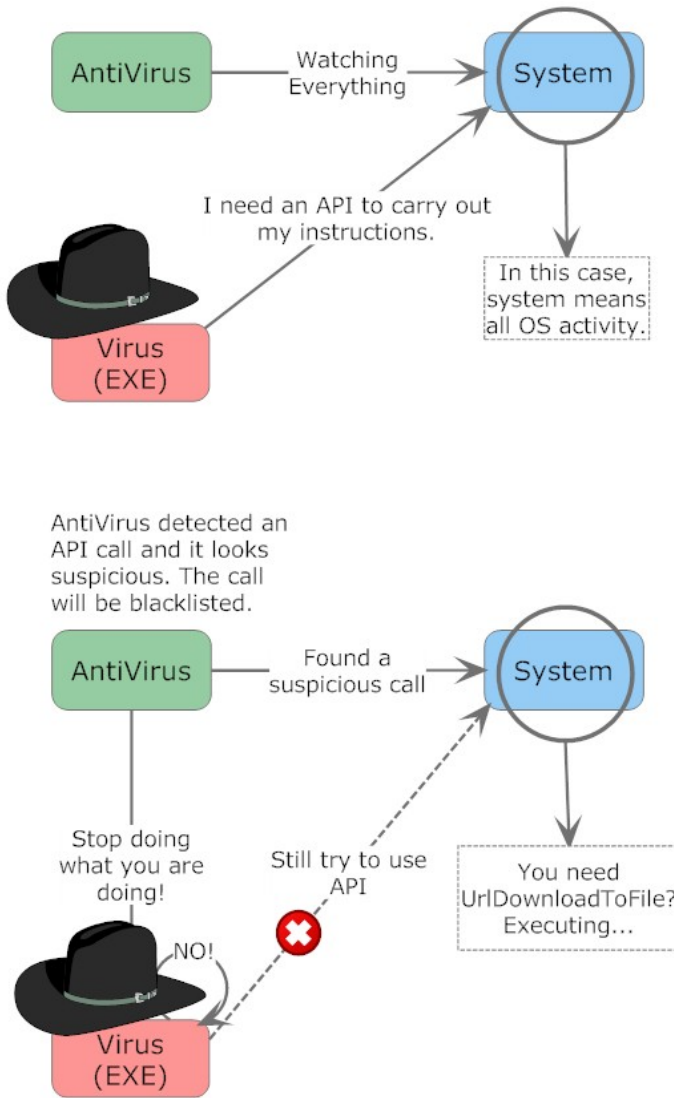


Figure 2.2: A botnet author's own description of what is perceived to happen under normal execution of exe files.

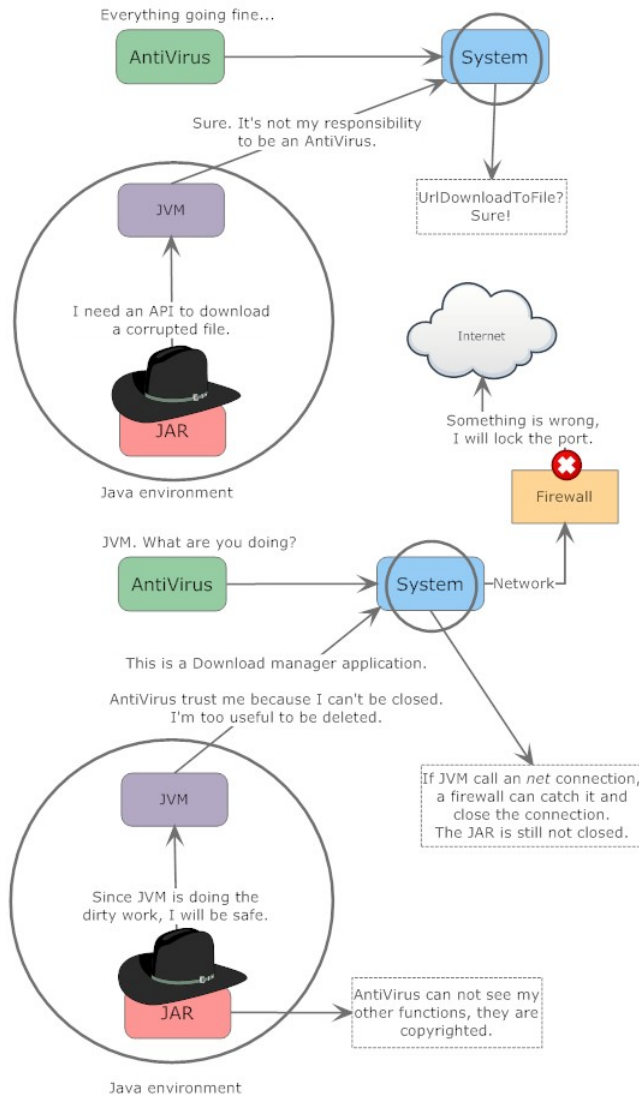


Figure 2.3: A botnet author’s own description of what is perceived to happen under normal execution of JAR files.

C2 topologies can typically be categorized into two main categories:

- **Centralized** C2 Botnets as shown in Figure 2.4, 2.5 and 2.6.
- **Decentralized**, or random, P2P Botnets as shown in Figure 2.7

### 2.5.1 Centralized topologies

The centralized command approach has been proved to be the most widespread with regards to botnet topologies. It provides the C2 botnet with a rendezvous point in the network, where the botnet administrator can coordinate malicious activity with the botnet. Within the category of centralized topologies, there are some sub categories:

- Star C2 topology
- Multi-server C2 topology
- Hierarchical C2 topology

#### Star C2 topology

The *Star* C2 topology is the simplest and arguably the most widely used topology for botnets, as shown in Figure 2.4. This topology relies on a single centralized C2 server for communication with all the bot agents. Once a new victim has been comprised, each bot will “phone home” to receive new instructions directly from the central command server. [67, p. 1-2]

*Pros: Speed of Control:* The direct connection between the C2 server and the bot agent enables fast transfer of instructions and stolen data.

*Cons: Single point of failure:* If the central C2 server is blocked or disabled, the botnet will effectively be neutralized. [67, p. 2] The bot may however still try to connect to the server.

**IRC-based Command and Control (C2):** Of the C2-based botnets, the most prominent would be the Internet Relay Chat (IRC)-based botnet. According to the Active Threat Level Analysis System (Atlas) summary report,

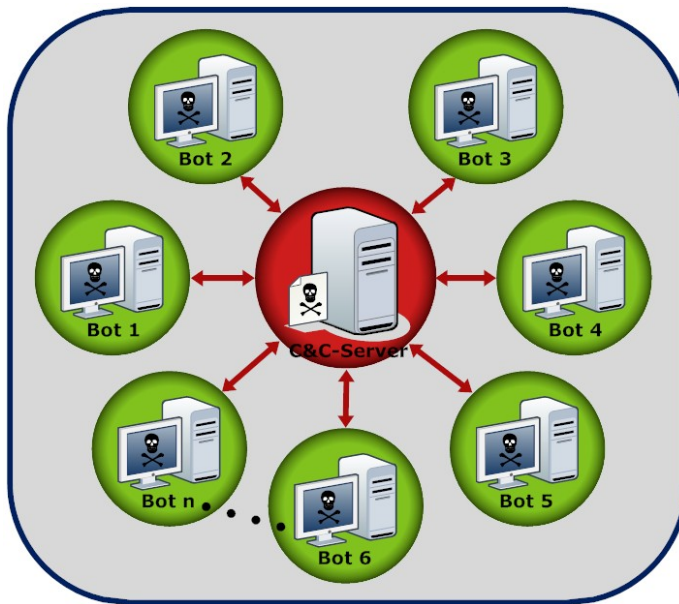


Figure 2.4: Star C2 topology with direct communication between the central command server and each bot agent. [67, p. 2]

per 21.03.2010 from Arbor networks [61], 56,7% of all server ports used was port 6667 which is the default port for IRC servers. <sup>1</sup>

These servers have been favored for their ease-of-use and they require minimal administration for C2. Botnet administrators can use public IRC channels or create their own. Private IRC-servers can be hosted by botnet friendly IRC hosting companies, run on previously compromised hosts with high bandwidth using dynamic Domain Name System (DNS) names for ease of access. *Dyn-DNS* [22] and *No-IP* [63] are some of the more popular choices for this type of service in conjunction with UnrealIRCd [86] as server software. [9, p. 9] This is a commonly used method of botnet administration according to the research that has been done. The IRC channel topic is often used to give instructions to the botnet or private messages to a specific bot if only one is needed. [62, p. 16-17]

Even though only IRC is mentioned here, there are many other protocols that also are in use for C2 botnets. Such protocols as *ICQ* [79], *Yahoo! Messenger* [94] and *MSN Messenger* [59] can be used to relay private messages to the bot.

**Web-based Command and Control (C2):** Another method of controlling a botnet is by HyperText Transfer Protocol (HTTP). Attackers can instruct the bot to access scripts on a web site and include its identifying information, such as the machine's IP address and what port its proxy is running on, with the Uniform Resource Locator (URL). This will ensure that the web-based C2 server can track and send commands to the entire botnet via HTTP responses. [62, p. 17-18]

With a Web-based C2 it is also possible to use social networking sites such as Twitter [20] or LinkedIn [30] for issuing commands to the botnet. Passing messages encoded with a shortened URL and hiding it in a picture has also proved to be a possibility, as shown with the proof-of-concept bot KreiosC2 [104]. This will make it much harder to detect the presence of botnets.

**DNS-based Command and Control (C2):** The Domain Name System (DNS) enables domain names such as "Twitter.com" to be translated into numeric IP addresses. By using a domain name as the rendezvous point, the botnet can make the C2 traffic look legitimate. One of the biggest advantages of this is that even if DNS queries are blocked by the firewall, the local DNS server could still forward the queries to an authoritative server and the C2 traffic would still pass though the firewall. [62, p. 20]

---

<sup>1</sup>As of 2010.06.07, 51.6 % of all server ports used were port 6667.

A special technique for preserving botnets is known as “fast flux DNS”. This is where the domain name’s Time-To-Live (TTL) is set to be very short. The consequence is that the IP addresses to that domain will constantly change and the lists that are used can range in the thousands per domain name. [24, p. 329]

### **Multi-server C2 topology**

*Multi-server* C2 topology is an extension of the Star topology where multiple servers are used to supply C2 instructions to the bot agents. The C2 servers communicate amongst themselves with regards to how they manage the botnet. Should one of the servers fail, the remaining servers would assume control of the bots so they are not lost. This topology requires more planning and effort when constructing the botnet infrastructure, but the bot agents can be used in the Star and Multi-server topology. By purposefully distributing the C2 servers across different geographical locations, communication with the specific bot can be sped up. If the servers are hosted in multiple countries it can also make it more resilient to legal shutdown requests. [67, p. 2-3]

*Pros:* **No single point of failure:** Should any of the C2 server be taken down or be disabled, the botnet administrator can still maintain control over all the bot agents. **Geographic optimizing:** Multiple geographically distributed C2 servers can speed up communications with the different botnet components.

*Cons:* **Require advance planning:** An additional preparation effort is needed to construct this multi-server C2 infrastructure. [67, p. 3]

Multiple servers can be achieved by adding backup IRC servers and channels if the bot is unable to contact the C2 server.

### **Hierarchical C2 topology**

A *Hierarchical* topology is in place when bot agents have the ability to proxy new C2 instructions to previously infected computers. This will help to control different parts of the botnet and it helps to hide the identity of the C2 rendezvous point and botnet administrator because of the additional communication layer. This will however introduce latency and make it harder for the botnet administrator to use the botnet in real-time. By using a hierarchical topology will also mean that no single bot will know the full extent of the entire botnet.



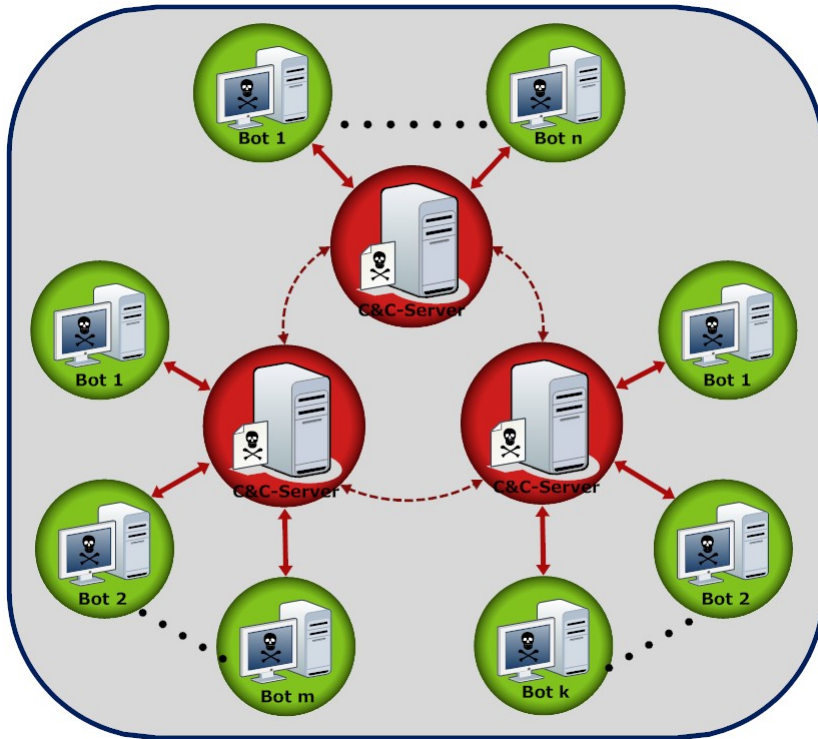


Figure 2.5: Multi-server C2 topology with direct communications between each distributed cluster of command servers and each bot agent. [67, p. 3]

This makes it harder for law-enforcement or investigators to estimate the true size of the botnet and it makes it easier for the botnet administrator to carve up parts of the botnet for re-sale. [67, p. 3-4]

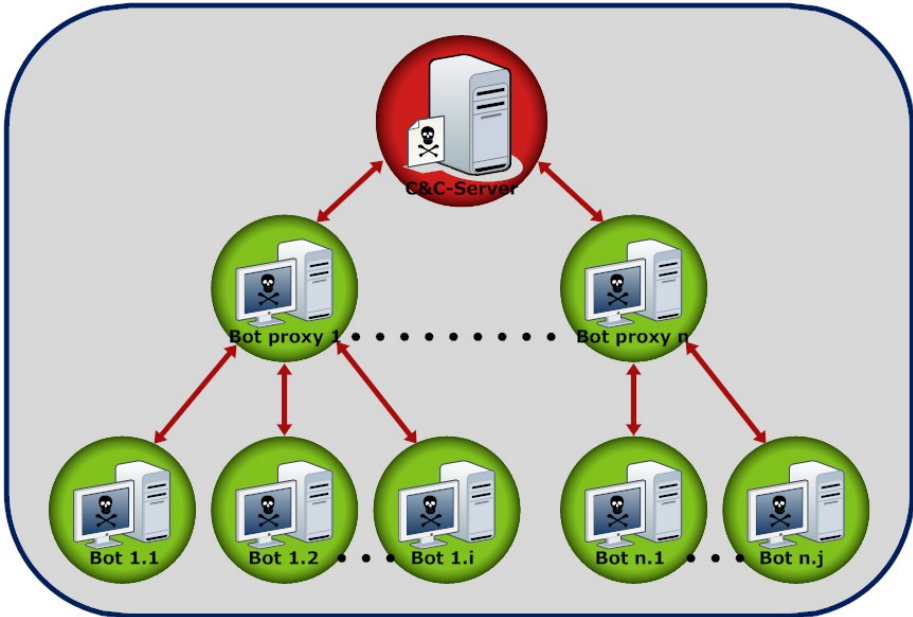


Figure 2.6: Hierarchical C2 topology with proxied C2 communication between the botnet administrator and bots. [67, p. 4]

*Pros:* **Botnet awareness:** Interception or hijacking of bot agents will not reveal all members of the botnet and will unlikely reveal the C2 server. **Ease of re-sale:** A botnet administrator can easily carve up parts of the botnet for sale or rent.

*Cons:* **Command latency:** Because there is no direct connection between the C2 server and the bot agent, there can be a high level of latency when updated instructions are being passed to the bot agents. This can make it difficult to execute real-time attacks. [67, p. 4]

## 2.5.2 Decentralized topologies

The *decentralized* command approach is a more recent botnet topology. There is no distinct C2 rendezvous point in the network and the botnet administrator must prove their identity to the network. The best analogy to this topology would be a Peer-to-Peer (P2P) architecture. Within the category of decentralized topologies, there is one main sub category:

- Random C2 topology

### Random C2 topology

A *Random* topology means that there is a dynamic master-slave or Peer-to-Peer (P2P) relationship between bot agents. There is no centralized C2 infrastructure, instead commands are sent to the botnet via any of the bot agents. These commands are then “signed” to ensure control and then propagated to all other agents. Due to the lack of a centralized C2 server and multiple communication lines between bot agents, the botnet is very resilient towards shutdown or hijacking. It is easy to identify other members of the botnet by observing a single infected host. There is a problem with latency because of the random topology, but with multiple communication lines this is less of a problem. [67, p. 4]

Sinit [82] and Phatbot [83] are examples of bots that have successfully taken advantage of this form of communication. Phatbot utilizes the Gnutella cache [16] servers to establish its list of seed peers and the compromised computers uses a modified version of the WASTE [90] protocol. [62, p. 19-20] The eDonkey protocol [35] has also been known to have been used for P2P-based botnets.

*Pros:* **Highly resilient:** Lack of a centralized C2 server and many-to-many communication between bot agents makes the botnet difficult to shutdown.

*Cons:* **Command latency:** The ad hoc nature of links between bot agents makes C2 communication unpredictable, this may result in a high latency for some parts of the botnet. **Botnet enumeration:** Passive monitoring of a single bot agent can enumerate other members of the botnet. [67, p. 5]

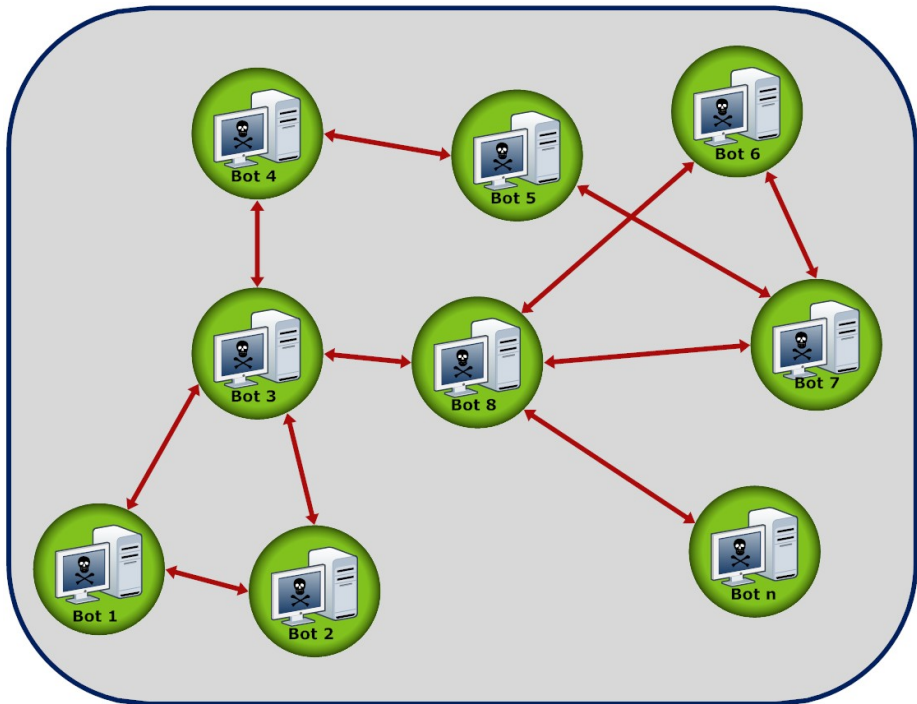


Figure 2.7: Random C2 topology with no centralized C2 server infrastructure. [67, p. 5]

## 2.6 Botnet metrics

Visualizations are useless without data or information from which to base the visualization. According to “A Host-Based Approach to BotNet Investigation?” [36] it is possible to detect botnets with a combination of host based data and packet captures in order to prove the presence of a botnet. I will take a further look into what host based data are available and its uses in the remaining sections of this chapter.

Botnets follow a Command and Control (C2) architecture and therefore they need to stay in contact with a server in order to get new commands and instructions. When communicating with the server it usually uses TCP/IP protocol for this task and the communication travels through the host computer and out onto the Internet. In its most basic form, data is stored in *ones* and *zeros*. But for a developer to be able to create more advanced applications, it is useful to abstract away this information. This will enable the development of more advanced applications and logic, and the developer can focus on the more difficult part of creating the logic.

### 2.6.1 Terminology

In order to have a common understanding of the terminology surrounding the data sources available for visualization, it is important to point out some key terms. These definitions are based on the Common Event Expression (CEE) standard that was published in 2007, but with slight modifications [12, p. 23, 30] [41, p. 22-23].

- *Event*: An *event* is observable situations or modifications within an environment that occurs over a time interval. An event may be a state change or reporting of an activity by a single component within a system.
- *Log Entry*: A *log entry* is a single record involving details from one or more events. It is sometimes referred to as an event log, event record, alert, alarm, log message, log record, or audit record.
- *Log*: A *log* is the collection of one or more log entries, typically written to a local log file, stored in a database or sent across the network to a server. A log may also be referred to as an audit log or audit trail.

- *Time-series data*: All data that can be attributed a point in time. An example includes log records with timestamps.
- *Static data*: Data that has no inherit time associated with it.
- *Parsing*: This is the process of taking a log or a file and identifying each part of it.

### 2.6.2 TCP/IP Model

The TCP/IP model, also known as the Internet model or DOD model, and its related protocols is a description framework, as shown in figure 2.8. It forms a complete system defining how data should be processed, transmitted, and received on a TCP/IP network. It is not as strictly divided into layers as the OSI model and it has four layers which include; *Application Layer*, *Transport Layer*, *Internet Layer* and *Network Access Layer*. A system of related protocols, such as the TCP/IP protocols, is also called a **protocol suite**. [10, p. 8]

The relation between the seven-layer of the OSI model and the four-layer TCP/IP model are shown in Figure 2.9.

#### TCP/IP packet headers

Each layer in the TCP/IP protocol stack performs specific duties. Each of these layers invokes services that are needed for that specific layer in order to complete its duties. As an outgoing communication travels down through the different layers, it includes a bundle of relevant information called a *header* along with the actual data. [10, p. 22] A detailed example of this can be seen in Figure 2.10.

From a metrics standpoint each of these layers represents valuable sources of information. In the context of botnet metrics the TCP and IP headers and their information is of the most importance. Due to the C2 nature of botnets, they need to be able to communicate with the rendezvous point. This communication is the same as every other communication traversing the Internet and therefore they rely on the same infrastructure. The TCP/IP protocol suite, or Internet Protocol Suite, forms the backbone of the Internet and Local Area Network (LAN). Some of the layers of the TCP/IP models layers such as the *Link Layer* and *Application Layer* does not provide information to others communicating parties unless the

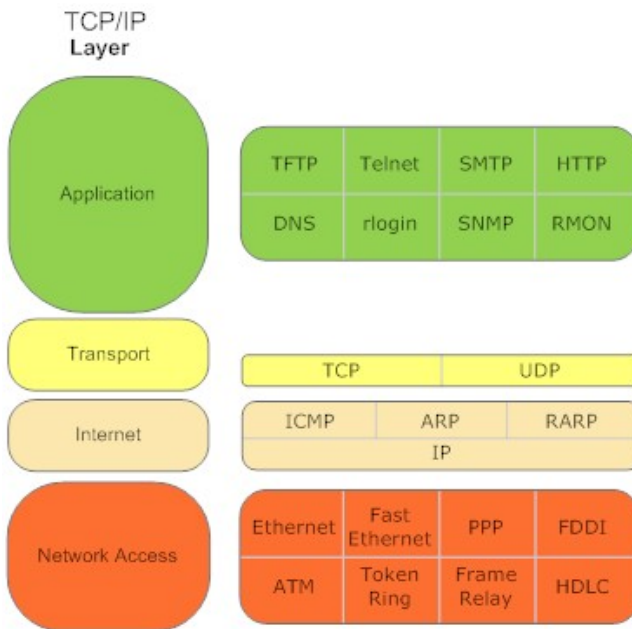


Figure 2.8: The TCP/IP model's four layers of abstraction show with typical protocols used in each layer.

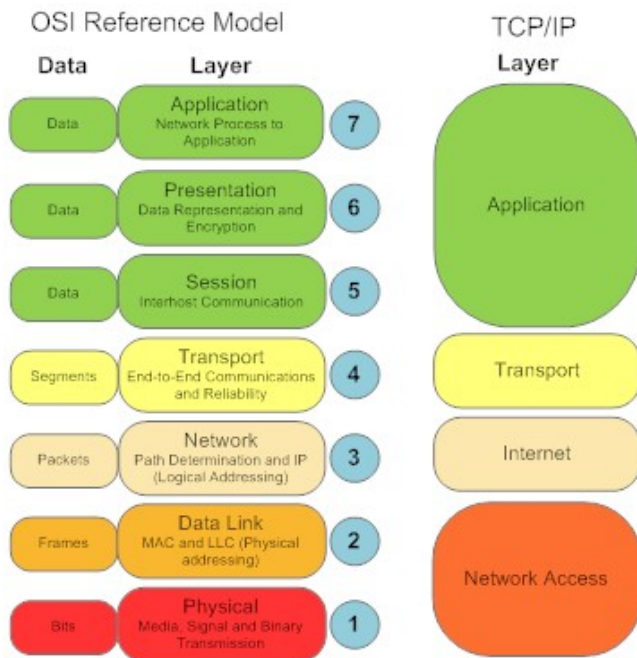


Figure 2.9: Differences between the OSI and TCP/IP model.





communicating computers are in the same LAN. If that is the case then an Address Resolution Protocol (ARP) can provide valuable information about the *Link Layer*.

With regards to the normal Internet communication between computers, the only layers that provide significant information to the other party are the *Network Layer* and the *Transport Layer*.

The TCP header from the *Transport Layer* and IP header from the *Internet Layer* provide metrics based on the knowledge that IRC-based botnets is C2 in nature and need a direct connection with a rendezvous point. TCP is a *connection-oriented* protocol and therefore it will be the natural communication choice.

**IP header fields:** The IP header includes a great deal of information including source and destination IP addresses of computers, Time-To-Live (TTL) and many others fields. I will focus on what I believe can provide interesting metrics in the context of this thesis. The header fields from figure 2.11 are: [70, p. 11-15]

- *Version:* Indicates whether the version of IP used is IPv4 or IPv6. Even though there is nothing in the IPv4 specification about geolocation (identification of a real-world geographic location), it is possible to deduce what city a given IP resides from, based only on the fact that IP address blocks tend to be allocated to regional Internet Service Provider (ISP)s. As of today it might be easier to determine geographical location of IPv4 addresses based on this fact. IPv6 based geolocation is out of the scope of this thesis.
- *Total Length:* Identifies the length of the datagram including IP header and data payload. This might be useful with regards to extracting the amount of information being transmitted. As botnet communication tend to be relatively short and contain a small amount of data compared to human communication, this may aid in classifying traffic. [85, p. 17]
- *Time-To-Live (TTL):* This field indicates the amount of time or how many **router hops** that the datagram can survive before being discarded. A *router hop* is a route the packet travels though on its way to its destination. This field is used actively by the *tracert/traceroute* commands mentioned in section 2.6.5.



- *Source Port*: This is the port number assigned to the communicating application on the host machine. This can help with connecting running processes on a host machine to the communication. If an “unknown” application is communicating then this can help indicate suspicious activity. Well known port numbers and what application is likely to be using them, is readily available and can aid in *process-to-port* mapping. [2]
- *Destination Port*: This is the port number assigned to the communicating application on the destination machine. This can help with establishing the type of application which the host application is communicating with. Well known port numbers and what application is likely to be using them, is readily available and can aid in *process-to-port* mapping. [2]
- *Sequence Number*: If the *SYN* flag is set to 1, this field provides the Initial Sequence Number (ISN) which is used for synchronizing. If the *SYN* flag is set to 0, then it provides the first byte of a particular segment.
- *Acknowledgment Number*: If the *ACK* flag is set to 1, this field contains the next sequence number that the receiver is expecting. This will be the *last sequence number + 1*.
- *Control flags*: The Control flags communicate special information about the segment. It indicates what stage a connection is in and contains: *URG*, *ACK*, *PSH*, *RST*, *SYN* and *FIN*. *URG* tells that the segment is urgent, *ACK* indicates if the *Acknowledgment Number* is set, *PSH* tells the TCP software to push data through the pipeline, *RST* resets the connection, *SYN* indicates if the Sequence number will be synchronized and marks the start of a connection and *FIN* is used to indicate that the sender has no more data to send.

The *Sequence Number*, *Acknowledgment Number* and *Control flags* tell us something about the connection and who initiated it. This can be valuable information as to indicate if the host computer contacted an external location, or if someone connected to the host computer without prior communication. It can also provide information about how many packets that are being sent, and we can then have an idea of the amount of information that has been transmitted. Thus one can choose to discard the remaining packets if we want to only focus on the initial connection.

**Example communication:** When two computers communicate over the Internet, information may be sent through many **router hops** along the way.



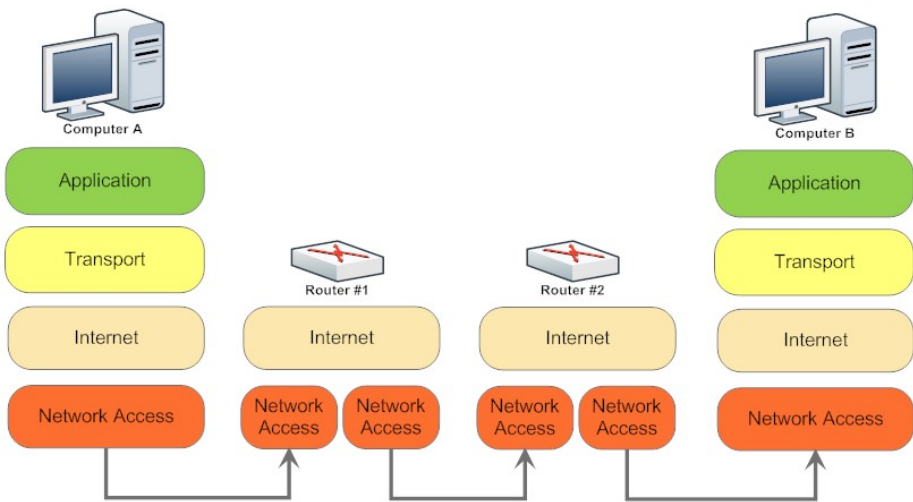


Figure 2.13: Example of routers that forward, but do not process Transport Layer data.

vantages: Firstly, there is no logic applied to the traffic in order to interpret it, hence we do not know how the destined application processes the information. Secondly, since the capture contains all the information within the packet it will produce vast amounts of data and in high volume networks this is impractical to analyze. This was a discussion topic in my meeting with UNINETT [27] where it was mentioned that their sample rate was 1 of every 100 packets due to the high traffic rate. Thirdly, there are legal and ethical problems with capturing the payload of a communication as it may contain private and sensitive information. [41, p. 27-28]

### **Legal and Ethical aspects**

Before discussing packet capture and analysis further it is important to keep the legal and ethical aspects in mind. An analogy to packet capture would be a postman opening or making a photocopy of all letters and then reading them. We would call opening someone's letters an invasion of privacy and this is also against the law. §122 [64] and §145 [65] of the Norwegian Penal code states that this is punishable by two to six years in prison depending on the severity. Packet captures could also be relevant with regards to §145 [65] as it includes attaining unauthorized access to data. We can look at packet captures as being relevant to the contents of letters or in this case the payload. Illegally collecting information could also be thought of as wiretapping if the captured packets include voice conversation or video conferencing. The point that was made in the meeting with UNINETT [27] and NTNU IT [28] was that it would be an invasion of privacy to analyze the payloads from packets flowing through their network. Hence, using payloads for metrics will not be included in this thesis.

Returning to the postman analogy it is also important to note questions raised by the sender and receiver address. In order to deliver mail and messages postmen need to know where to send the message and so there is a need for the destination address to be known to ensure delivery. If the sender also wants their letter to be returned, in case the recipient does not receive their mail, they can add the sender address to the mail. This is a well accepted practice, this changes when it comes to Internet traffic. As presented earlier in this thesis, one can still get vast amounts of personal information with only the source and destination addresses. In the meeting with UNINETT [27] they pointed out the sensitive subject of collecting IP addresses as this might be seen as an invasion of privacy. The point of the meeting was to investigate what kind of information

that could be used for metrics.

Although UNINETT wanted to ensure the privacy of their users, not everyone follows this practice and great debate has been sparked by the Swedish FRA law [69] and possible Norwegian implementation of the EU data retention directive. [17] The FRA law would enable Sweden's National Defense Radio Establishment (FRA) to monitor all outgoing and incoming communications crossing Sweden's borders in the name of national security. This would in practice make wiretapping of all Internet traffic through Sweden a reality. [69] I will not go into too much detail about these topics, but it is important to realize that the legality of what can be stored in which circumstance is subject to interpretation.

### Packet capture tools

There is a variety of tools available to collect and analyze network traffic. The most common ones are Wireshark [93], Tcpdump [88], WinDump [97] and NetworkMiner [29]. Other tools bundled with Wireshark [93] include dumpcap [91] and tshark [92]. These tools listen on the network interface and take the raw network traffic and analyze the entire packet to decode the individual network protocols. Whereas Wireshark [93] and NetworkMiner [29] provides a graphical user interface to better analyze the packets, dumpcap [91], tshark [92], Tcpdump [88], and WinDump [97] are command-line tools only.

These tools all use the PCAP format from libpcap/Tcpdump [88] which is the most common format for storing network captures. Wireshark [93] is not designed for very large network captures so dumpcap [91], Tcpdump [88] or WinDump [97] are more suited for this purpose. When using Tcpdump [88] or WinDump [97] there are some settings that are important to mention: [41, p. 28]

- *-s 0*: By default WinDump [97] only captures the first 68 bytes of the packet. This is enough to include IP, ICMP, TCP and UDP headers. By setting this parameter you will capture the packet headers and the payload. [98] [89]
- *-n*: Setting this parameter will prevent WinDump [97] from converting addresses (i.e., host addresses, port numbers, etc.) into hostnames. [98] [89]
- *-e*: Setting this parameter will print the link-level header on each dump line. This will typically be the Media Access Control (MAC) addresses.



- `-w filename.pcap`: This will enable writing the output to a PCAP file.

The actual data contained within a PCAP [88] file-capture is set to one packet per line. The data is stored as bytes and thus it is not human readable, but using the following command it is possible to see what type of information is contained in a typical capture:

```
WinDump.exe -n -e -i 2 -s 0
```

A sample PCAP [88] packet from the previous command yields:

```

❶19:39:40.528361 ❷00:13:21:c9:8b:26 > ❸00:0c:cf:32:48:00,
ethertype IPv4 (0x0800), length ❹85:
❺129.241.208.163. ❻443 > ❼80.249.84.43. ❽3989: ❾P
❿2653951672:2653951691(19) Ⓜack
Ⓜ3408490598 Ⓜwin 64400
Ⓜ<nop,nop,timestamp 3000146 1659722>

```

The following list shows the typical information you can extract from a packet capture. For more information about the specific fields in this list, reference Section 2.6.2:

- *Timestamp* ❶: By default all output lines are preceded by a timestamp of the current clock in the form: `hh:mm:ss.frac` This is dependent on the kernel clock and thus are only as accurate as the kernel clock. [98] [89]
- *MAC address* ❷: When the `-e` parameter is set, the MAC address is displayed. This can reveal information about the local network. [98] [89]
- *Packet Length* ❸: The packet length of the packet. [98] [89]
- *IP addresses* ❹: The IP addresses of the endpoints in this communication. [98] [89]
- *Ports* ❺: The ports used by the communicating IP addresses. [98] [89]
- *TCP Flags* ❻: The Control flags in the TCP header. *P* in this case indicates the *PSH* flag is set. The *ack* indicates that this packet contains an *acknowledgment number*. [98] [89]
- *Sequence Number* ❼: The sequence number of the TCP packet and it indicates that has data. [98] [89]

- *Acknowledgment Number* ⑨: The acknowledgment number of the TCP packet. [98] [89]
- *Window* ⑨: win 64400 tells that one can transmit only 64,400 bytes of data before getting a frame extension. [98] [89]
- *Options* ⑩: Data within these brackets describe options. [98] [89]

In some cases it is preferable to analyze more than the TCP/IP packets. Higher-level protocols such as IRC and instant messenger traffic are not shown with Tcpdump [88] or WinDump [97]. Wireshark [93] and its bundled command-line application tshark [92] can be useful in this circumstance. Before demonstrating the tshark with a sample packet dump it might be useful to mention some tshark settings. By first using WinDump [97] for the packet capture and then use tshark [92] to analyze the capture:

- *-t a*: This sets the timestamp of the packet capture to the actual time the packet was captured instead of the default format, which is relative. [92]
- *-r filename.pcap*: Will read packet data from an input file. [92]
- *> filename.txt*: Will direct the output to a text file. [92]

A sample capture analyzed with tshark showing the possibility to extract user names from instant messenger and IRC traffic. Identifying information has been masked by **XXXXXX**.

```
21:10:33.111285 129.241.208.163 -> 157.158.2.161 TCP
http > 54613 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
MSS=1460 WS=8 TSV=3545401 TSER=3527938244
```

```
21:10:33.111450 129.241.208.163 -> 157.158.2.161 TCP
telnet > 52865 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

```
21:10:34.470212 129.241.208.79 -> 129.241.209.255 UDP
Source port: 17500 Destination port: 17500
```

```
21:10:34.632333 129.241.208.163 -> 217.17.33.10
IRC Request
```

```

21:10:40.927979 65.54.172.104 -> 129.241.208.163 MSNMS
NLN BSY 1:XXXXXXXXXX@hotmail.com
Name 2788999228:402822192 <msnobj Creator="
XXXXXXXXXX@hotmail.com" Type="3"
SHA1D="HYbhW4luo864gdjU7qSMhy3h6e0=" Size="10192"
Location="0" Friendly="SwBvAGsAawAAAA=="/>

```

## 2.6.4 Source/Destination Confusion

This is a term presented by Raffael Marty in the book “Applied Security Visualization” and I will present the example from the book to clarify this issue. [41, p. 26-27] If you assume that you are recording network traffic. Here are two packets from an HTTP connection, recorded by Tcpdump. [88]

```

❶ 18:46:27.849292 IP 192.168.0.1.39559 > 127.0.0.1.80: S
1440554803:1440554803(0) win 32767
❷ 18:46:27.849389 IP 172.0.0.1.80 > 192.168.0.1.39559: S
1448343500:1448343500(0) ack 1440554804 win 32767

```

The connection was made to the loopback interface to access the Web server that is running on the host laptop. If a simple parser is used to extract the source and destination IP addresses, you end up with a graph that looks like Figure 2.14



Figure 2.14: Simple graph showing the concept of source/destination confusion.

It is possible to see that there are two arrows, one from ❶ 192.168.0.1 to 127.0.0.1 and one in the ❷ other direction. The connection was however only from 192.168.0.1 to 127.0.0.1, and no connection was opened the other way around.

According to Marty, this happens because the parser mechanically extracted the source addresses as the first IP in the log, without trying to understand which packets were flowing from the client to the server and vice versa.

This is very important because in a botnet perspective we want to know if our computer was connecting to a remote server or if the remote server was connecting to us. This could indicate either a botnet trying to connect to its rendezvous point or the botnet administrator might attempt to gain access to the system. To solve this problem Marty presents the need for a smarter parser that keeps track of the individual clients and server pairs. Afterglow [40], which was written by Marty, includes the file *tcpdump2csv.pl* which he claims can solve the source/destination confusion problem. It handles this problem by using the *SYN* and *SYN ACK* parts of the three-way-handshake to keep track of who started the communication.

Regardless of whether a solution based on Marty's parser is implemented, it further points out the significance of the TCP *Control Flags* from Section 2.6.2.

## 2.6.5 Operating System (OS) metrics

Information useful for creating metrics for detection and visualization of botnets does not stop with network traffic and packet captures, but the OS has logs that can provide useful information. In Chapter 2 botnets was explained and as mentioned previously, botnet applications are the same as every other applications running on a host computer; an application. The botnet may try to employ techniques such as using a rootkit, mentioned in Chapter A, to hide from the OS logs. This will help avoid detection, but they still have an impact on the system.

To get a better idea of what tools are available for gathering metrics from the host machine in the fight against botnets, I will give a short list of the most common and useful tools for this purpose.

### Network and task management tools

Some of the network management commands/tools that are included with UNIX systems can help with providing host based data is *nslookup/dig*, *traceroute/tracert*, *ping*, *ipconfig/ifconfig/winipcfg*, *netstat* and *Windows*

*Management Instrumentation Command-line (WMIC)*. These tools come with the operating system and can be accessed through a command shell.

- *nslookup/dig*: These commands are used to query DNS for resource records. This may include translating a hostname, such as *www.example.com*, into its corresponding IP address, which is *192.0.32.10*, and vice versa. On large sites such as *www.google.com* this might result in multiple IP addresses and these might change over time for a number of reasons. This property of being able to have multiple IP addresses mapped to a single domain and change these addresses over time is a DNS technique, called *Fast Flux*. This is often exploited by botnet administrators to hide C2 servers. [24, p. 329] [51]
- *tracert/tracertpath/tracert*: These commands are used to determine the most likely route taken by packets traveling through a network. When the Time-To-Live (TTL) field in an IP packet reaches zero and the packet did not reach its destination the router that is handling the packet will respond back to the sending host with information about where the packet was dropped. [54]
- *ping*: This command is used to determine if a host is reachable on the network. Sometimes referred to as ICMP ping because of its use of the Internet Control Message Protocol (ICMP), this is part of the core protocols of the Internet Protocol Suite 2.6.2. [52]
- *ipconfig/ifconfig/winiipcfg*: These commands are used to show all current network interface configurations, such as current IP, the *Default Gateway* and *MAC address* of the network adapter. By using `ipconfig /displaydns` it is also possible to view the resolver cache for DNS on the host machine [49].
- *netstat*: This is a Windows command that provides information about network statistics, open sockets and other network related information. [50] This can be a very useful tool for matching running processes to communicating ports when used in conjunction with another built in Windows tool called *tasklist* [53].
- *WMIC*: This is a command-line tool in Windows that allows access to system management information such as running processes on the system. It can provide similar information to *netstat*, but also vast amounts of other

system information. [55] By running `wmic startup list full` a detailed list of what applications that run at startup is shown, which might give information about suspicious applications running in the background. Running `wmic process get processid, Name, executablepath` will give information about the running process.

Although these tools provide useful information about network traffic and running applications, it is still *static data* (see Section 2.6.1). Another problem with these tools are that they don't provide log files of the information, but instead you will have to regularly run the commands and parse the data to be able to store them in log files. WMIC does provide functionality to output data to Comma-separated values (CSV), but the command still needs to be executed regularly. A good collection of Windows management tools can also be found on Microsoft's Sysinternals website. [76] [41, p. 48-52] A note about these applications is that a botnet with *rootkit* capabilities, as explained in Chapter A, can hide itself from these applications. Thus, there is a risk that botnets are not detected when using these tools.

## SNMP

Simple Network Management Protocol (SNMP) is a protocol for acquiring data automatically from network management systems. It is part of the Internet Protocol Suite, as explained in Section 2.6.2, and is used to monitor devices. The protocol is based on scalar technology and simple definition of managed objects in which data is stored. [87, p. 44, 105] [11, p. 5]

The management information is stored in the Management information base (MIB) and it is a type of virtual information store (base). The MIB contains a (virtual) database of hierarchical objects and are obtained with the use of object identifier (OID)s, where the OID is the objects position in the MIB [87, p. 109, 157, 159, 181] [46, p. 6] [47] For UNIX system there is a library called Net-SNMP [80] available that allows access to SNMP information. [41, p. 50]

OID's that can be used to extract useful host based network management information include:

- *.1.3.6.1.2.1.25.4.2.1.1*

```
.iso.org.dod.internet.mgmt.mib-2.host.
hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunIndex
```

Displays a unique value for each piece of software running on the host, which is the Process IDentifier (PID) of the running application. [100]

- .1.3.6.1.2.1.25.4.2.1.2

```
.iso.org.dod.internet.mgmt.mib-2.host.
hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunName
```

This displays the *Name* of the running application. If this software was installed locally, this should be the same string as in the corresponding hrSWInstalledName. [100]

- .1.3.6.1.2.1.25.4.2.1.3

```
.iso.org.dod.internet.mgmt.mib-2.host.
hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunID
```

This displays the *product ID* of the running application. [100]

- .1.3.6.1.2.1.25.4.2.1.4

```
.iso.org.dod.internet.mgmt.mib-2.host.
hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunPath
```

This displays the *Path* of the running application. It provides a description of the location on long-term storage, for example a disk drive, from which this software was loaded. [100]

- .1.3.6.1.2.1.25.4.2.1.5

```
.iso.org.dod.internet.mgmt.mib-2.host.
hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunParameters
```

Displays a description of the parameters supplied to the running application when it was initially loaded. [100]

- .1.3.6.1.2.1.25.4.2.1.6

```
.iso.org.dod.internet.mgmt.mib-2.host.  
hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunType
```

Displays what type of software the running application is. Values can be *unknown*, *operatingSystem*, *deviceDriver* or *application* [100]

- *.1.3.6.1.2.1.25.4.2.1.7*

```
.iso.org.dod.internet.mgmt.mib-2.host.  
hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunStatus
```

This displays the status of the running application. Values can be *running*, *runnable*, *notRunnable* or *invalid* [100]

- *.1.3.6.1.2.1.25.6.3*

```
.iso.org.dod.internet.mgmt.mib-2.host.  
hrSWInstalled.hrSWInstalledTable
```

This displays a list of installed software on the host computer with its installed *name*, *type* of software/application and *date* of install. [100]

- *.1.3.6.1.2.1.6.13.1.1*

```
.iso.org.dod.internet.mgmt.mib-2.tcp.  
tcpConnTable.tcpConnEntry.tcpConnState
```

This displays the state of this TCP connection. [75]

- *.1.3.6.1.2.1.6.13.1.2*

```
.iso.org.dod.internet.mgmt.mib-2.tcp.  
tcpConnTable.tcpConnEntry.tcpConnLocalAddress
```

This displays the local *IP address* for this TCP connection. In the case of a connection that is in the listen state, which means that it is willing to accept connections for any IP interface associated with the node, the value 0.0.0.0 is used. [75]

- *.1.3.6.1.2.1.6.13.1.3*



```
.iso.org.dod.internet.mgmt.mib-2.tcp.
tcpConnTable.tcpConnEntry.tcpConnLocalPort
```

This displays the local *port number* for this TCP connection. [75]

- .1.3.6.1.2.1.6.13.1.4

```
.iso.org.dod.internet.mgmt.mib-2.tcp.
tcpConnTable.tcpConnEntry.tcpConnRemAddress
```

This displays the remote *IP address* for this TCP connection. In the case of a connection that is in the listen state, which means that it is willing to accept connections for any IP interface associated with the node, the value 0.0.0.0 is used. [75]

- .1.3.6.1.2.1.6.13.1.5

```
.iso.org.dod.internet.mgmt.mib-2.tcp.
tcpConnTable.tcpConnEntry.tcpConnRemPort
```

This displays the remote *port number* for this TCP connection. [75]

- .1.3.6.1.2.1.6.19.1.1

```
.iso.org.dod.internet.mgmt.mib-2.tcp.
tcpConnectionTable.tcpConnectionEntry.tcpConnectionLocalAddressType
```

This displays the address type of the local *IP address* for this TCP connection. [75]

- .1.3.6.1.2.1.6.19.1.2

```
.iso.org.dod.internet.mgmt.mib-2.tcp.
tcpConnectionTable.tcpConnectionEntry.tcpConnectionLocalAddress
```

This displays the local *IP address* for this TCP connection. [75]

- .1.3.6.1.2.1.6.19.1.3

```
.iso.org.dod.internet.mgmt.mib-2.tcp.
tcpConnectionTable.tcpConnectionEntry.tcpConnectionLocalPort
```

This displays the local *port number* for this TCP connection. [75]

- *.1.3.6.1.2.1.6.19.1.4*

```
.iso.org.dod.internet.mgmt.mib-2.tcp.  
tcpConnectionTable.tcpConnectionEntry.tcpConnectionRemAddressType
```

This displays the address type of the remote *IP address* for this TCP connection. [75]

- *.1.3.6.1.2.1.6.19.1.5*

```
.iso.org.dod.internet.mgmt.mib-2.tcp.  
tcpConnectionTable.tcpConnectionEntry.tcpConnectionRemAddress
```

This displays the remote *IP address* for this TCP connection. [75]

- *.1.3.6.1.2.1.6.19.1.6*

```
.iso.org.dod.internet.mgmt.mib-2.tcp.  
tcpConnectionTable.tcpConnectionEntry.tcpConnectionRemPort
```

This displays the remote *port number* for this TCP connection. [75]

- *.1.3.6.1.2.1.6.19.1.7*

```
.iso.org.dod.internet.mgmt.mib-2.tcp.  
tcpConnectionTable.tcpConnectionEntry.tcpConnectionState
```

This displays the state of this TCP connection. [75]

- *.1.3.6.1.2.1.6.19.1.8*

```
.iso.org.dod.internet.mgmt.mib-2.tcp.  
tcpConnectionTable.tcpConnectionEntry.tcpConnectionProcess
```

The system's Process Identifier (PID) for the process associated with this connection. This is expected to be the same as *hrSWRunIndex*.

With the use of SNMP it is possible to retrieve the network information that I believe is necessary to help detect botnets. Another aspect of the use of SNMP for collecting this information is that it is designed to function in a distributed context. This means that if I manage to use these host-based metrics to detect the presence of a botnet, then this solution can be implemented in a larger scale. Microsoft included SNMP functionality as standard with Microsoft Windows XP/Windows 2000/Windows NT and later versions. [56]

According to Aiko Pras, Associate Professor at the University of Twente and member of Internet Research Task Force (IRTF) Network Management Research Group, the reason why *tcpConnectionEntry* gives “empty” answers in Microsoft Windows is that the objects are not implemented since it is fairly new:

“Anyway, it is often the case (for example for all old devices we can access via the simpleweb [19]), that certain MIB objects are not yet implemented. Then you get an “empty” answer.

Unfortunately our Simpleweb [19] devices are all relatively old, and do not support the new *tcpConnectionTable*. I would not be surprised if this is true as well for many devices currently on the market.”—Aiko Pras, e-mail 2010.05.18 & 2010.05.19

What this means for this thesis is that the *process-to-port* mapping cannot be done entirely in SNMP and therefore might be difficult at this time to implement in a distributed context. Netstat will therefore be used for this purpose. As with netstat, SNMP on Microsoft Windows may be vulnerable to botnets with *rootkit* capabilities. If a bot takes control over the system files associated with SNMP, it can feed false information back to the user. [57] If the *process-to-port* mapping is carried out in a distributed context within an organization that has access to traffic data, one can compare the results from a central collection point with the results from the SNMP agent. If the data collected by the central point does not correspond to what the SNMP agent returns, then this can indicate a problem with that particular host machine.

## 2.7 Simple analysis of a Java Based bot

To weave together the information introduced in this and previous chapters, I will do an analysis of two real world bots. These are both Java based and are

very simple and generic examples of bots. It is unlikely that they pose a threat to networks, but they do have IRC communication features that provide a real world example on how botnets communicate. I will use the simpler one of the two latter for the experiment after I have disabled any potentially dangerous and damaging code blocks.<sup>2</sup>

It seems like these two botnets come from the same source, but have been further developed by different people. I call the first Java bot “Family Photos” because of the filename it is saved as.<sup>3</sup> According to the author this was to prevent the file-uploading service from deleting the file. It builds the code into a JAR file named *jusched.jar* which functions as the complete bot application. *Jusched.exe* is the name of “Java Update Scheduler” and this bot uses a filename similar in an attempt to hide from the user. A screen shot of the botnet source can be seen in Figure 2.15. The second Java bot I call “IRC” based on the filename it is saved as.<sup>4</sup> The JAR file with the bot application is also called *jusched.jar*, which added to my suspicion that the bots come from the same source.

The focus will be on the “Family Photos” bot but similar or added functionality from the “IRC” bot will be mentioned. It will be clearly stated which bot is referenced. An analysis of the “IRC” bot has been written by Craig Williams, which was also mentioned in Section 2.4, and for a detailed analysis one should reference his Cisco blog post. [103]

### 2.7.1 Botnet setup

The botnet provides customizable variables in order to setup the botnet. The location of the IRC server to be used with the botnet and which port it should use must be provided. It uses separate IRC channels for command input and bot response output, each with fields for the password needed to gain access to the channels. Prefix for the bot’s nickname and what nicknames are designated the botnet administrators must also be provided.

A list of the variables to be set is extracted from the *ConfigDefaults.java* file are as follows:

---

<sup>2</sup>Both bots mentioned along with the modified version will be included on the DVD.

<sup>3</sup>“Family Photos” source code location per 20.06.2010: <http://www.megaupload.com/?d=VC72061M>

<sup>4</sup>“IRC” source code location per 20.06.2010: <http://uppit.com/TW69DX>

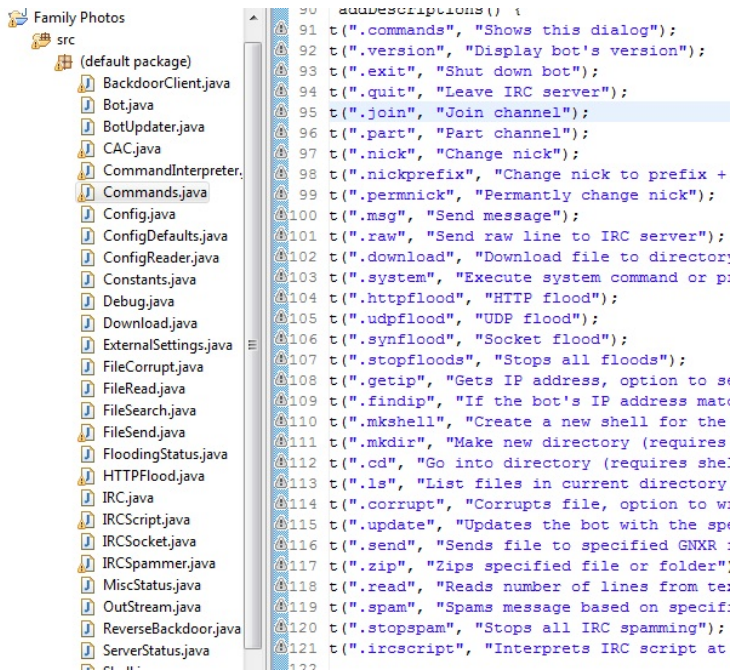


Figure 2.15: Excerpt of the available botnet commands for the “Family Photos” bot.

- *SERVER*: YOURIRC.SERVER.COM
- *PORT*: 6667
- *IN\_CHANNEL*: #SomeBotnet
- *OUT\_CHANNEL*: #SomeBotnet2
- *CHANNEL\_IN\_PASSWORD*: lolz
- *NICK\_PREFIX*: [BOT]
- *CHANNEL\_OUT\_PASSWORD*: lolz
- *CONTROLLERS*: IRCUser1,IRCUser2,IRCUser3

Before the configuration variables are loaded, the bot first tries to establish if a remote configuration file is available. Preferences are loaded from *Constants.java*. If there is no external configuration file, the bot then loads variables from *ConfigDefaults.java*. The issue with the code in particular is that if you don't have a valid URL from which to find a configuration file, it will throw an exception and crash the bot. The *URLChecker.java* takes an URL as input, but if there is no valid URL it throws an unhandled `java.net.MalformedURLException`. The same thing will happen if the backup URL is not valid.

The nicknames listed in the *CONTROLLERS* variable are the nicknames of the botnet administrators. It does not carry out commands from other nicknames than these.

This bot corresponds to the Star C2 topology described in Section 2.5.1, but by adding the possibility for backup IRC servers it would be considered to be a Multi-server C2 topology, which is described in Section 2.5.1

### Botnet setup for “IRC” bot

In addition to similar variables as the “Family Photos” bot, the “IRC” bot has added variables to set password protection for the bot in order to prevent other competing botnet administrators from stealing it. It also has File Transfer Protocol (FTP) support and the ability to use a remote setup file.

## 2.7.2 Botnet startup

At startup of the “Family Photos” bot, the main class “Bot.java” is run. It reads the configuration file and starts a new C2 instance to an IRC server on default port 6667. After the connection has been made the bot adds itself to startup. Depending on whether the OS is Windows XP or Windows Vista the bot adds itself one of the following paths respectively, where “userHome” is the system property “user.home”:

```
userHome + "/Start Menu/Programs/Startup"
```

```
userHome + "/AppData/Roaming/Microsoft/Windows/Start Menu/  
Programs/Startup"
```

To see whether the bot could successfully be added to startup I did a test on the host machine. Due to access restrictions set by Windows to the /Start Menu/Programs/Startup/ folder, the file was not able to be added to the startup. It was compiled and put in the C:\Users\Andreas\ folder.

Once the bot has added itself to startup, it starts itself to run in the background. Finally it builds a new jar file named Syn.jar from byte code. Since the classes are built from byte code and can only be viewed after it has compiled, they are shown in Appendix D.1 and D.2. Syn.jar contains two classes: “a.class” and “Main.class”.

### Botnet startup for “IRC” bot

At startup the bot reads its configuration file and attempts to connect to an IRC server on default port 6667. Once a connection to a server is made, it attempts to add itself to startup and the registry startup:

```
HKLM\Software\Microsoft\Windows\CurrentVersion\Run
```

The “ProcessChecker” class will run after the bot is added to startup. This is explained in Section 2.7.4.

### 2.7.3 Botnet commands

The botnet commands communicate the capabilities of the bot, and these include IRC channel specific commands, downloading remote files, executing files on the host, flooding capabilities, spam capabilities, ability to transmit files and bot commands for update and shutdown among others. These commands are directly extracted from the *Commands.java* file:

- *.commands*: “Shows this dialog”
- *.version*: “Display bot’s version”
- *.exit*: “Shut down bot”
- *.quit*: “Leave IRC server”
- *.join*: “Join channel”
- *.part*: “Part channel”
- *.nick*: “Change nick”
- *.nickprefix*: “Change nick to prefix + random number”
- *.permnick*: “Permanently change nick”
- *.msg*: “Send message”
- *.raw*: “Send raw line to IRC server”
- *.download*: “Download file to directory”
- *.system*: “Execute system command or program”
- *.httpflood*: “HTTP flood”
- *.udpflood*: “UDP flood”
- *.synflood*: “Socket flood”
- *.stopfloods*: “Stops all floods”
- *.getip*: “Gets IP address, option to set URL to read from”
- *.findip*: “If the bot’s IP address matches, it says so”



- *.mkshell*: “Create a new shell for the channel”
- *.mkdir*: “Make new directory (requires shell on the channel)”
- *.cd*: “Go into directory (requires shell on the channel)”
- *.ls*: “List files in current directory (requires shell on the channel)”
- *.corrupt*: “Corrupts file, option to write message”
- *.update*: “Updates the bot with the specified jar file URL”
- *.send*: “Sends file to specified GNXR file server on specified port”
- *.zip*: “Zips specified file or folder”
- *.read*: “Reads number of lines from text file”
- *.spam*: “Spams message based on specified settings”
- *.stopspam*: “Stops all IRC spamming”
- *.ircscript*: “Interprets IRC script at specified URL”

When the command *.getip* is called, the bot uses an automated service from *whatismyip.com* [101] to determine its public IP address. This feature, along with the *.commands*, *.exit*, *.quit*, *.join*, *.part* and *.raw* commands provide all the features necessary to carry out experiments with regard to detection of botnets based on communication. This will be explored further in Chapter 4.

### **Botnet commands for “IRC” bot**

The “IRC” bot has similar commands, but also some added features. Whereas the “Family Photos” bot use “.” to indicate a commands, this bot uses “\$” instead. Since the command file is different from the previous one and do not contain descriptions so I will add them. The commands are extracted from the *CommandProcess.java* file and I show only those that are added features:

- *\$unlock*: Unlocks the commands of the bot if the supplied password is correct.

- *\$lock*: Lock the commands of the bot if the supplied password is correct.
- *\$getUsername*: Return the username of the logged in user on the host machine.
- *\$takeScreenshot*: Take a screenshot of the host computer.
- *\$getHomeDir*: Return the home directory of the logged in user on the host machine (“user.home” system property).
- *\$getOS*: Return the OS name running on the host machine (“os.name” system property).
- *\$processList*: Return a list of running processes with the use of “tasklist”.
- *\$uploadFile*: Upload a file by FTP.
- *\$disableProcessChecking*: Disable process checking.
- *\$enableProcessChecking*: Enable process checking.

#### 2.7.4 Summary for bot analysis

Both bots use IRC for communication on the default port 6667. This corresponds to the Star C2 topology, described in Section 2.5.1, which mentions that the majority of botnets use port 6667 and IRC servers for communication.

In order for the bots to load at startup they need to register with the system to make sure the compromised computer is not lost. Both bots can also add themselves to startup, but “IRC” bot also adds itself to the registry. This was also mentioned in Chapter A.

To investigate the claim that a Java based botnet is FUD the *jusched.jar* was submitted to VirusTotal [78], which is a service that uses over 40 different antivirus engines to analyze suspicious files for the presence of viruses, worms, Trojan horses, and all kinds of malware. The results of this scan are shown in Figure 2.16.

As the results from VirusTotal [78] shows, the antivirus engines did not detect the bot. This seems to confirm the claim that a Java based botnet is FUD. The date of this first scan also shows the age of the bot to some extent. This goes to show that this bot have been around for some time.

**File jusched.jar received on 2009.06.30 19:56:51 (UTC)**  
**Current status: finished**  
**Result: 0/41 (0.00%)**

Figure 2.16: VirusTotal scan results for *jusched.jar*

The *ProcessChecking.java* class in the “IRC” bot is quite interesting because it looks for what the bot deems as “bad” processes running on the system and then executes a command to kill the blacklisted processes. Once enabled, it kills off *tcpview.exe*, *wireshark.exe*, *etherreal.exe* and *netstat.exe* in an attempt to hide itself from the user. The method is run every 30 seconds to reduce system impact, but it is unlikely it would cause too much problems if netstat was used for data collection. This is because of the relatively large waiting time between runs. Hiding from system applications was mentioned in Section 2.2, where bots have tried a variety of ways to hide themselves and from *netstat.exe* in particular. Bots also use “rootkit” technology, as mentioned in Chapter A, to hide themselves from the system, but neither of the bots use this technology. The fact that botnet authors try to kill these applications just goes to show that they are seen as a threat to the bot.



# Chapter 3

## Visualization

“Ordinary people believe only in the possible. Extraordinary people visualize not what is possible or probable, but rather what is impossible. And by visualizing the impossible, they begin to see it as possible.”—Cherie Carter-Scott

### 3.1 Introduction

In order to display vast amounts of information to the user in a easy and understandable manner one can make use of visualization tools. Instead of manually monitoring vastly different log files and try and extract the useful information, we try and extract the essence of what is happening and present it to the user. Computers are good at following rules and *pattern recognition* is one example of this. According to C. M. Bishop:

“The field of *pattern recognition* is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories.” [7, p. 1]

What seems to be the common factor in pattern recognition is the focus on machine learning with a training set, where the training set is known in advance and manually inspected. A big focus is on mathematics and discovering patterns by looking for clustering or use of probability theory. [7, p. 2-4] When visualizing botnets this might not be the only information that might be useful. Humans seem to be better equipped to see patterns in data and respond when something unexpected happens. Pattern detection, especially with regards to geography, seems to be an intuitive skill for humans.

A visual representation of data enables the communication of vast amounts of information to the users in a clear and compact way. To do this in the best possible way it is important to know something about how people understand visuals. An experiment performed by Tolman [95] on “Cognitive maps in rats and men”, led to the concept of the *cognitive map*. This is where one develops a mental map of an area. There have been many notions for mental representation of spatial knowledge and among these are *spatial images* by Kevin Lynch, which will be discussed later in this section. [3, p. 22]], and work by Barkowsky to create a model for representing and processing lean geographic knowledge in the human mind. It further points to geography as an important factor when making sense of the world. [3, p. 65]

In the book “Applied Security Visualization” by Marty [41], a number of different possibilities for visualizing data are presented including different charts, histograms, plots, graphs, maps and tree maps. 3 of the 13 proposed visualization methods focused on geographical visualizations and only one directly referenced a geographical context with event information. [41, p. 110-113] The focus seems to be on traffic volume instead of events and this may indicate an area where it is possible to contribute.

In the book, *The Image of the City* [38] written by Kevin Lynch, a five year study was conducted on how people perceive and organize spatial information as they navigate through cities. Using three cities which were “vivid in form and full of locational difficulties” [38, p. 14], Lynch reported that people understood their surroundings in consistent and predictable ways, forming mental maps with five elements:

1. *Paths*: Paths include streets, sidewalks, railroads or other paths where people observe their surroundings while moving.
2. *Edges*: Edges include walls, shores, and edges of development or barriers.

ers. They are linear elements not considered to be paths, but breaks in continuity.

3. *Districts*: Districts are relatively large sections of a city with a common identifying character. For example: “Business district”.
4. *Nodes*: Nodes are points of focus or reference such as intersections, places of a break in transportation, crossings or convergence of multiple paths. The person can enter these nodes such as driving through an intersection.
5. *Landmarks*: Landmarks are typically physical objects such as a building, sign, store or mountain. They are another point of reference, but in this case the person does not enter them.

The study found that paths were the predominant city element, although this varied in importance according to their familiarity with the city. People who had little knowledge of the city tended to think in terms of topography, large regions, generalized characteristics, and broad directional relationships. People who knew the city best of all relied more upon small landmarks and less upon regions or paths. [38, p. 46-49]

## 3.2 Visualization metrics

To find good visualization tools we need to know how to accommodate as much as possible of Lynch’s five elements in order to create an intuitive visualization tool. Since it is nearly impossible to have intimate knowledge of vast amounts of log data and packets, *Paths* and *Edges* should be the main focus of the tool in question. However, to accommodate more advanced users and to help maintain the “big picture” it is also important to have some reference to the other elements.

By taking cues from the five elements and the concept of the *cognitive map* at the start of this chapter, I believe that mapping data in a geographical context would not introduce new reference systems to the user. By providing metadata in a compact manor we can reduce the amount of trivial details. This will cover the *spatial dynamics* aspect of the problem. To address the *temporal dynamics* aspect we also want the visualization to change over time without too much interaction from the user. It is also important that it is easy to make changes to the visualization as experimentation introduces many changes.

The focus is not on the volume of traffic, but on the connection itself as the information being sent is not as important as the fact that the computer is communicating with a C2 rendezvous point in the first place. This means that there is no linear mapping of connections onto the visualization. By overwriting each node when subsequent connections to the same IP are captured, it will reduce the amount of nodes even if the activity of a particular connection increases. This will help keep focus on the initial connection and not on how active or how much data is being transmitted.

Each node or connection will be grouped and assigned an image icon that corresponds to the application responsible for the connection. Mapping of the application to the connection will be performed with *process-to-port* mapping as described in the previous chapters. The executable's name will be presented to the Visualization application in order to assign and compare the image icon. This will aid in spotting irregularities, emphasize the difference between connections and show the footprint of an application. For example: If the majority of the connections are from a web browser, an IRC connection will stand out based on the different image icon.

Once an interesting observation or connection has been made, it must be possible to extract more information from the particular observation. Interesting information such as IP, port, hostname of the server in question, described in Chapter 2, as well as location information such as city or country will be presented in the visualization itself without having to leave the application. This will improve the usability because a user can perform a quick analysis before deciding to investigate manually instead of having to investigate each suspicious connection.

Since the focus of the thesis is not detection, but visualization of botnets a simple scheme of white listing applications known to the user will be employed. This will be done based on assigned image icons for the applications such that an unknown application will be presented to the user for further investigation.

### 3.2.1 Geolocation

Geolocation is the identification of TCP/IP traffic or other computer metrics to real-world geographic, latitude/longitude coordinates or Universal Transverse Mercator (UTM) [21] spatial address grid coordinates. When trying to project locations onto the earth's surface one may face challenges with regards to



projecting the round earth onto a flat map.

## Map projection

The Mercator *projection* is a conformal projection which means that the angles and shapes on the globe, project as the same angles or shapes on the map. What this means in practice is that there is a great variation in scale when you move away from the central portions of the map. This means that even though Greenland has a surface area of 1/8 the size of South America it is portrayed as being the same size. [21] There are numerous ways of doing map projection, but the one I will use in the thesis is a geographic projection (Plate Carrée), also called *equiangular projection*, used by NASA in their *Blue marble* project. [60] It is based on equal latitude-longitude grid spacing. [84, p. 9] An example of this projection and the image used for the visualization can be seen in Figure 3.1

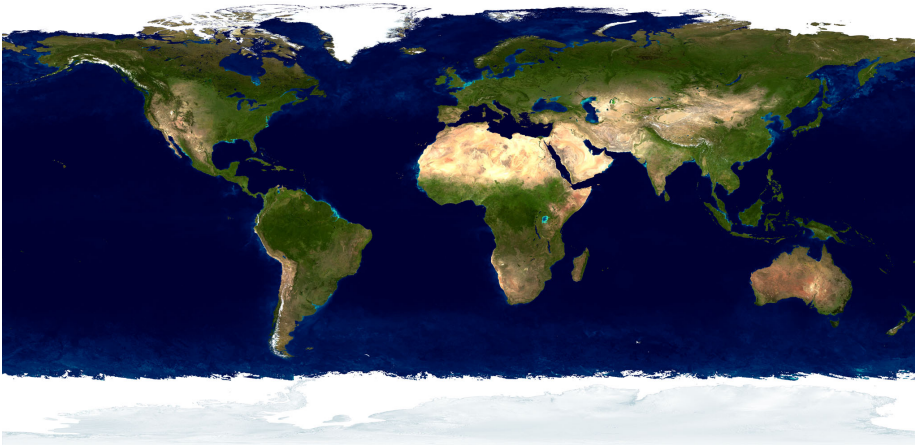


Figure 3.1: A world map projection from the NASA Blue marble project. [60]

## Geolocation tools and resources

A number of tools exist that provide data for geolocation and a number of vendors provide databases for translating IP addresses into latitude/longitude. The paper by Robert Lee and Sheau-Dong Lang noted the importance of locality [37] and by utilizing these resources; I believe a more informative visualization will be presented to the user. I will provide a quick introduction to some of the most important tools and vendors of geolocation databases.

- *Whois*: The *whois* service is the Internet equivalent of the phone directory. It provides network information, administrative, and technical contact information for Internet domains and IP addresses. [96, p. 299]
- *traceroute/tracert*: This *traceroute* utility is provided on all UNIX- and Windows-based systems to give the administrator the ability to show the most probable route the packets will use the target IP. It achieves this by using the TTL field, as mentioned in Section 2.6.2. Each time the TTL field reaches 0, the router handling the packet returns a time-expired ICMP error message to the original sending computer. By incrementing the TTL by 1 each time it receives an error message until it reaches the destination computer. It is possible to determine the most probable route the packet must have taken to the destination computer based on these error messages. The devices don't necessarily provide geographic information, but they are configured by ISP's that are geographically located. Having the location included in the host name helps them when a network issue requires troubleshooting. [96, p. 306-308]

Some of the best known database vendors for translating IP addresses to latitude/longitude are:

- *MaxMind*: MaxMind provides IP address geolocation for credit card fraud detection. They provide both paid and free solutions for determining a visitor's country or city of origin based on the IP address. [42]
- *IP2Location*: IP2Location provides services to help identify users based on geographical locations, i.e. country, region, city, latitude, longitude and other metrics based on IP addresses. [32]

- TorIp2Country: *Tor* [73] is an application to preserve a user’s anonymity online and it has a free public available database of IP address to country. [74]

The decision to use Maxmind’s *GeoLite City* database for this thesis was based on the quality of the product and because the price was free. Since the price of IP2Location’s DB18 product is \$1399 per server each year and the focus of the thesis is not geolocation in itself, the Maxmind database would suffice since accuracy was not of paramount importance. Testing has shown no missing or misplaced locations.

One note is that the Maxmind’s *GeoLite City* only supports IPv4 and not IPv6. There is however a *GeoLite IPv6 Country* database available.

## Geolocation limitations

The accuracy of geolocation must also be kept in mind when using these services and applications. If the target IP is routed through for example a satellite link, proxy server or the *Tor* [73] anonymity service, these factors may affect the accuracy of the results.

## 3.3 Technology

### 3.3.1 Processing

“Processing is an open source programming language and environment for people who want to program images, animation, and interactions. It is used by students, artists, designers, researchers, and hobbyists for learning, prototyping, and production. It is created to teach fundamentals of computer programming within a visual context and to serve as a software sketchbook and professional production tool.”—processing.org [5]

The project was originally developed by Ben Fry and Casey Reas, but has a large following of users that develop libraries for use with the software. A large

user following also creates new material on a regular basis and most is released openly to the community [72].

Features of Processing:

- Open source Java Software Development Kit (SDK).
- Extensive library support.
- Processing code is based on Java and can be used with any Java application.
- Large community backing.
- High quality visuals.
- Highly extendable and customizable.

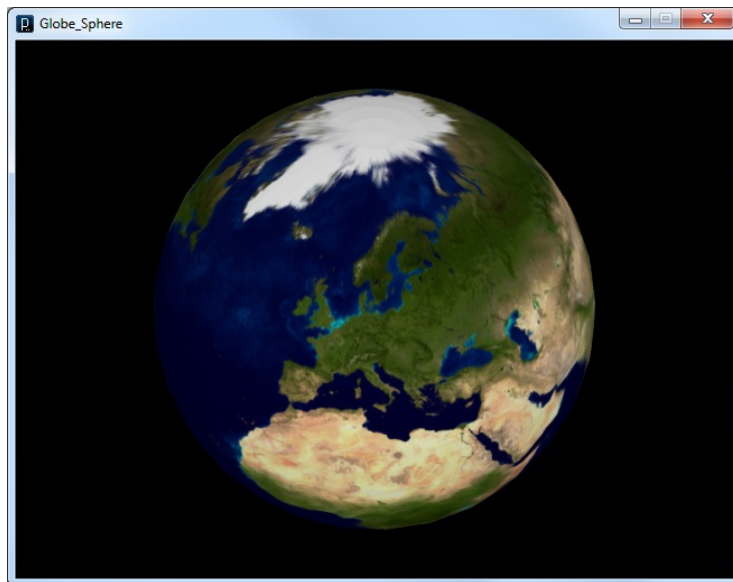


Figure 3.2: The rendered 3D representation of the Earth with Processing.

I was able to generate a 3D representation of the Earth, as shown in Figure 3.2, with only 129 lines of code extracted directly from an example on the

Processing [5] [58] website and with a picture from Blue Marble [60], shown in Figure 3.3.

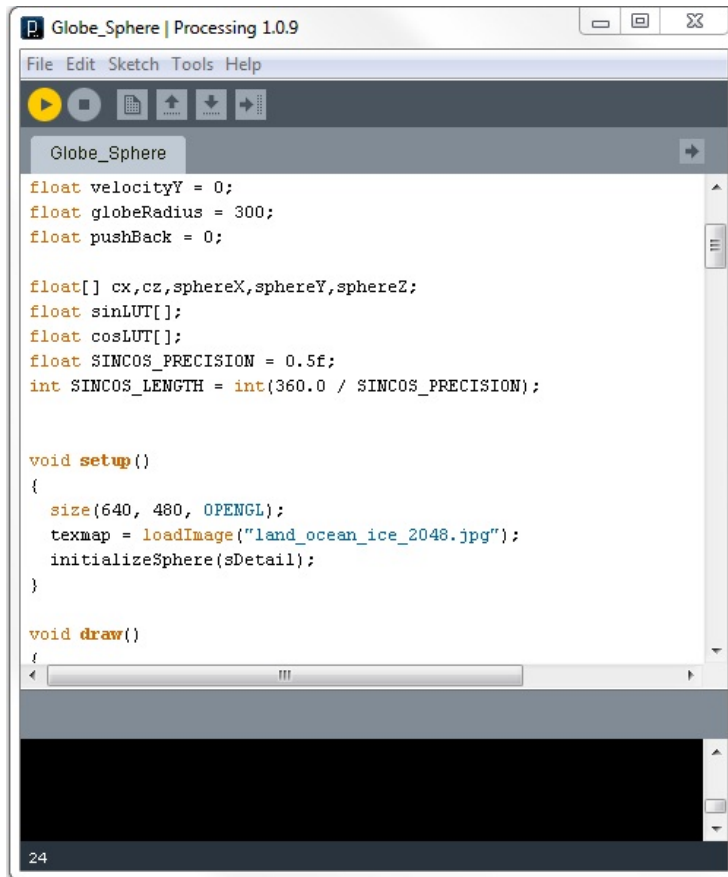
The processing library seems ideal to develop visualization applications as it is fairly easy to develop relatively elaborate visuals. With the use of a Java library for data collection one can easily integrate it into Processing.

### **Processing library - Carnivore**

“Carnivore is a surveillance tool for data networks”–Carnivore [23]

This is a library for processing which uses a network dump, captured by jPacp [13], to visualize the packets that flow through the network. The data is displayed based on their IP address and which port that is in use. The main feature here is that the application uses jPacp [13] to present the data stream in a controlled manor to the Processing application. It also features a way of displaying that information with icons and colors. This is highly customizable and an interesting option. An example of the carnivore library in action can be seen in Figure 3.4.

This library will provide a tested solution for packet captures and with some added functionality for visualizing the packets and collecting host based metrics, there should be sufficient material available for visualizing botnets.

The image shows a screenshot of the Processing IDE window titled "Globe\_Sphere | Processing 1.0.9". The window has a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for play, stop, save, copy, paste, and zoom. The main area is a code editor with a tab labeled "Globe\_Sphere". The code is as follows:

```
float velocityY = 0;
float globeRadius = 300;
float pushBack = 0;

float[] cx,cz,sphereX,sphereY,sphereZ;
float sinLUT[];
float cosLUT[];
float SINCOS_PRECISION = 0.5f;
int SINCOS_LENGTH = int(360.0 / SINCOS_PRECISION);

void setup()
{
  size(640, 480, OPENGL);
  texmap = loadImage("land_ocean_ice_2048.jpg");
  initializeSphere(sDetail);
}

void draw()
{
```

The bottom of the window shows a dark 3D rendering area, which is currently black. The status bar at the bottom left of the window displays the number "24".

Figure 3.3: Only 129 lines of code can generate a high quality 3D representation of earth in processing.

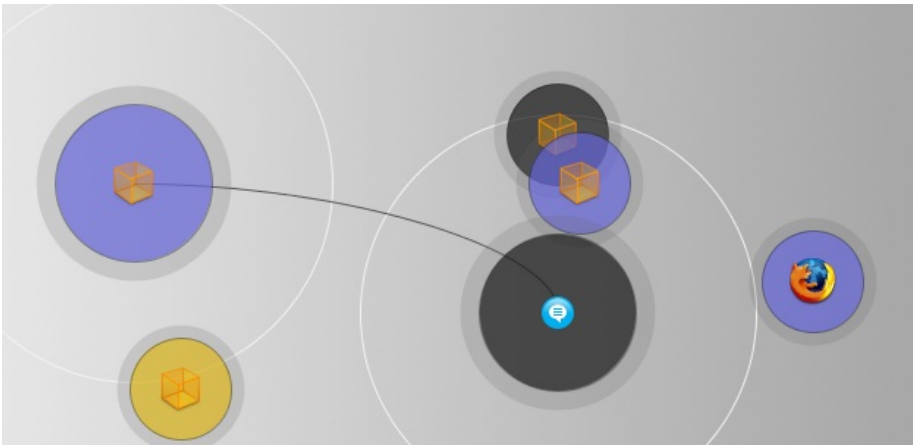


Figure 3.4: The Carnivore Library for Processing, displaying a packet dump. [23]





# Chapter 4

## Experiment

“The Internet is the first thing that humanity has built that humanity doesn’t understand, the largest experiment in anarchy that we have ever had.”–Eric Schmidt

### 4.1 Introduction

An important note to consider before carrying out this experiment is that one has to obtain permission from the parties involved. Not doing so may cause you to inadvertently break laws and ethical boundaries. For this experiment, the NTNU Information Technology (IT) department was briefed about the experiment and what was going to happen, the IRC server administrator was asked for permission with regards to carrying out the experiment and the bot in question had its attack features deactivated. The computers used in the experiment were setup by the author of this thesis and was borrowed with permission from NTNU. Only one bot was used for this experiment in order to limit the appearance of a botnet. Channel topics and other bot administrator information were set in such a way that a potential onlooker did not confuse the experiment with an actual live botnet. By not gaining permission from the IRC server administrator may cause them problems and could ultimately result in the server being shut down by their ISP.

All possible precautions were taken to avoid such a case and to keep the experiment within the boundaries of the law. By taking these precautions, the definition of a botnet, from Chapter A, will no longer apply for the bot involved in the experiment, but still keeping the communication capability so the experiment would be valid. Anyone wanting to replicate the experiment must keep in mind the legal and ethical aspects involved and perform the experiment in a safe and controlled manor.

## **4.2 Materials and experiment setup**

In this section a detailed description of the materials and preparation needed to carry out the experiment will be described.

### **4.2.1 Equipment**

The equipment needed for this experiment is the following:

- Two computers.
- Two Windows 7 licenses.
- Windows SNMP agent available for install.
- Wireshark [93] application.
- mIRC [39] client application.
- WinDump [97] application.
- CamStudio [14] application.
- Network connectivity.
- Network cables.
- Two keyboards.
- Two mice.
- Access to an IRC server.

- Access to two IRC channels.
- Java based IRC-bot source code.
- Visualization application.

## 4.2.2 Experiment Setup

The basic experiment setup consists of two computers located in Norway, which will serve the purpose of the bot, named “PC1”, and a bot administrator computer, named “PC2”. A remote IRC server located in South Africa will serve as the Command and Control (C2) rendezvous point for the two computers.

The bot controller will set up and monitor the IRC channels defined in the setup file of the modified “Family Photos” bot and issue some standard commands to the bot in order to ensure it is responding correctly. WinDump [97] packet capture will be performed on both machines and once this is done, the Visualization will be activated on the machine infected by the bot. Video documentation will be performed on both the bot and bot administrator computers with the use of CamStudio [14].

The basic setup of the experiment can be seen in Figure 4.1

## 4.2.3 Setup for IRC server and channel

The setup of the IRC server and the two IRC channels are done in accordance with guidelines set by IRC server administrators. They are configured to safely carry out the experiment in a controlled and transparent manor.

### IRC server

The IRC server that was used is located in Northcliff (Johannesburg), South Africa, and was chosen purely because of its geographic distance from Norway.

IRC server:

`za.webcomm.co.za`

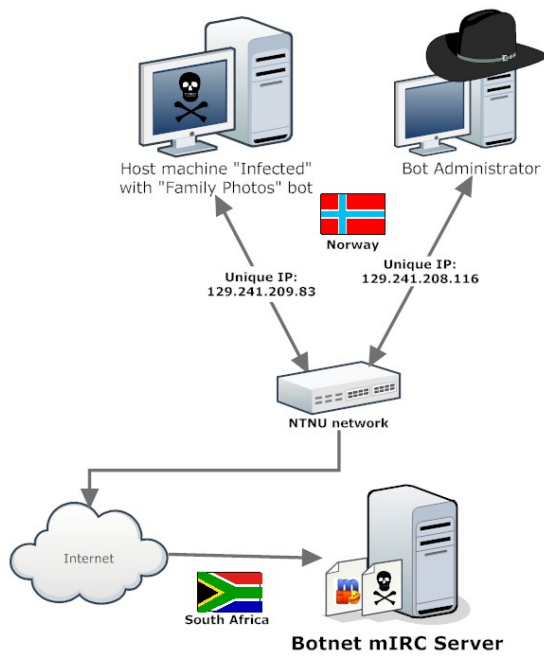


Figure 4.1: Basic setup for the experiment.

Permission was obtained from the IRC server administrators and they were briefed as to what the experiment consisted of. The administrators agreed to the experiment on two conditions:

1. The bot had to join the channel *#bots*
2. The bot had to set the mode: *MODE [TEST]-FIXED +B*, where *[TEST]-FIXED* was the nickname of the bot and *+B* is used to ban a nickname.

Doing this would point out to any onlookers that the bot was registered with their system and that they were informed of its presence. The bot commands necessary to complete these requirements are:

- *.join #bots*
- *.raw "MODE [TEST]-FIXED +B"*

### **IRC channel setup**

The IRC channels needed to be set up in an informative way to onlookers, as well as keeping the experiment as contained as possible. The channel modes used to achieve this is as follows [33] [39]:

#### **Channel modes used:**

- *+t <topic>*: Only channel operators may set the channel topic.
- *+n*: No external messages can be sent to the channel from users not in the channel.
- *+k <key>*: Keyword-protected which will prevent users to connect to the channel if not familiar with the password.
- *+p*: Private means that the channel is not visible in any listing.
- *+s*: Secret means that the channel can show up in *channellisting*, but users outside the channel cannot see users inside the channel.

When the channels were created they were set up as follows:

**Channel: #PC2\_Network\_in:**

- *Channel name:* #PC2\_Network\_in
- *Channel topic:* “Thesis Experiment IN”
- *Keyword:* “in1234”
- *Other channel modes:* +n, +p and +s.

**Channel: #PC2\_Network\_out:**

- *Channel name:* #PC2\_Network\_out
- *Channel topic:* “Thesis Experiment OUT”
- *Keyword:* “out1234”
- *Other channel modes:* +n, +p and +s.

#### 4.2.4 Modification of the “Family Photos” bot

For this experiment, a modification of the “Family Photos” Java based bot source, explained in Section 2.7, will be performed to neutralize its capabilities. This can be done since the complete source code is available. Because of ethical and legal considerations as well as rules from the server that was used, any damaging code blocks or potentially dangerous capabilities has been removed or deactivated. This would ensure that the bot used in the experiment did not cause damage or could be used as part of a real botnet.

As described in the analysis, from Section 2.7.2, the main class of this bot is the *Bot.java* class. Modification to the *ConfigReader.java* has also been done for the bot to work properly. With the current setup, as described in Section 2.7.2, the bot will crash if there is no valid external configuration file URL. This has been circumvented to further prevent the bot going rouge or terminating unexpectedly.

To point out that the bot had been stripped of its capabilities, all responses to the command: “commands” had “neutralized” appended to the response. Functionality for adding the bot to startup through *StartupAdder.java* and the running of *Syn.java* was also deactivated. In addition to these measures all classes except; *Bot.java*, *CAC.java*, *CommandInterpreter.java*, *Commands.java*,

*Config.java*, *ConfigDefaults.java*, *ConfigReader.java*, *Constants.java*, *Debug.java*, *IRC.java*, *IRCsocket.java*, *OutputStream.java*, *URLChecker.java* and *Util.java* have been removed.

In order for the bot to compile with the included binaries the *PATH* environment variable in Windows needs to be added:

```
C:\Program Files\Java\jdk1.6.0_17\bin
```

Once the bot is compiled it will create a jar file named *ModifiedJavaBot.jar*. This can be run from a command shell by issuing the command:

```
java -jar ModifiedJavaBot.jar
```

An antivirus scan on the compiled *ModifiedJavaBot.jar* bot was performed with the use of VirusTotal [78] and the results are shown in Figure 4.2. This shows that the bot was not detected and thus still have the same FUD properties as the antivirus results from the analyzed bot, from Section 2.7.4.

File **ModifiedJavaBot.jar** received on **2010.05.31 21:06:25 (UTC)**  
Current status: **finished**  
Result: **0/40 (0%)**

Figure 4.2: VirusTotal scan results for *ModifiedJavaBot.jar*.

### Modification of configuration defaults

The *ConfigDefaults.java* class was modified to reflect the IRC server to be used in the experiment. For more information on the configuration defaults of the bot, reference Section 2.7.1.

- *SERVER*: za.webcomm.co.za
- *PORT*: 6667
- *IN\_CHANNEL*: #PC2\_Network\_in

- *OUT\_CHANNEL*: #PC2\_Network\_out
- *CHANNEL\_IN\_PASSWORD*: in1234
- *CHANNEL\_OUT\_PASSWORD*: out1234
- *NICK\_PREFIX*: [TEST]
- *CONTROLLERS*: PC2Controller, PC2Controller2

The *IRCSocket.java* class is responsible for setting the nickname of the bot. As default it uses the *NICK\_PREFIX* variable along with a random number in order to generate the used nickname. This random number function was replaced with “FIXED” so the final nickname of the bot would always be: “[TEST]-FIXED”.

### **Modification of available commands**

The *Commands.java* class was modified to only contain the options described in Section 2.7.3 and are presented in detail:

- *.commands*: “Shows this dialog (Neutralized)”
- *.exit*: “Shut down bot (Neutralized)”
- *.quit*: “Leave IRC server (Neutralized)”
- *.join*: “Join channel (Neutralized)”
- *.part*: “Part channel (Neutralized)”
- *.raw*: “Send raw line to IRC server (Neutralized)”
- *.getip*: “Gets IP address, option to set URL to read from (Neutralized)”

### **4.2.5 Setup for PC1, a bot infected computer**

A clean installation of Windows 7 and latest critical updates were performed. User Account Control (UAC) was deactivated and the Windows Firewall was active. Wireshark [93], WinPcap [99] and CamStudio [14] were installed according to



the default installation settings provided by the applications. The Windows SNMP agent was then installed and configured according to the description in Appendix C.

In order for the Visualization to be run, the Processing [5] application needed to be installed on the computer and the infected computer will run a copy of the modified “Family Photos” bot, described in Section 4.2.4.

#### 4.2.6 Setup for PC2, a bot administrator computer

A clean installation of Windows 7 was performed and latest critical updates were installed. User Account Control for windows was deactivated and Wireshark [93], WinPcap [99], CamStudio [14] and *mIRC* was installed according to the default installation setting provided by the applications.

##### Configuration of mIRC [39] client

The mIRC [39] client needs to set some details before one can connect to an IRC server. This is done in the “mIRC Options Window”.

###### **Category: Connect settings:**

- *Full Name:* PC2
- *Email address:* none
- *Nickname:* PC2Controller
- *Alternative:* PC2Controller2
- *Invisible mode:* Activated (Tacked off)

###### **Category: Connect, Identd settings:**

- *User ID:* PC2Control
- *System:* UNIX
- *Port:* 113

**Category: Connect, Servers settings:** Finally, the server described in Section 4.2.3, *za.webcomm.co.za*, is added to the IRC server list and marked as selected.

## 4.3 Methods

Both computers have been denied Internet access to prevent them being contaminated by other malware, until the experiment was carried out. Before the experiment was performed, both computers were updated with the latest critical Windows updates and the firewall was activated. To describe the experiment in the greatest detail possible, a description of setup on each computer is shown. An extended description of the experiment is also shown in Figure 4.3.

Since there are multiple computers involved in this experiment, each step will be described for PC2 and PC1 respectively. It is important to carry out the experiment in this order, because the IRC channels should be set up beforehand and in order to log the bot's activity from the botnet administrator.

### 4.3.1 Step by step guide for PC2

The instructions and tasks to be carried out for the bot administrator computer is listed in sequential order. The purpose of these instructions is to make sure the instructions are carried out correctly and provide solid results to base the analysis on.

1. It is important to have the IP address of the computer for later analysis in order to correctly identify the machines involved. This is obtained by running *ipconfig*, described in Section 2.6.5, in a command shell.
2. Start WinDump [97] in a command shell with the following command: `WinDump.exe -n -e -i 1 -s 0 -w c:\controller.pcap` This is useful for analyzing what information is exchanged between the computers. For more information on WinDump, see Section 2.6.3.
3. Start mIRC and setup the client and channels according to Section 4.2.6 and Section 4.2.3 respectively.

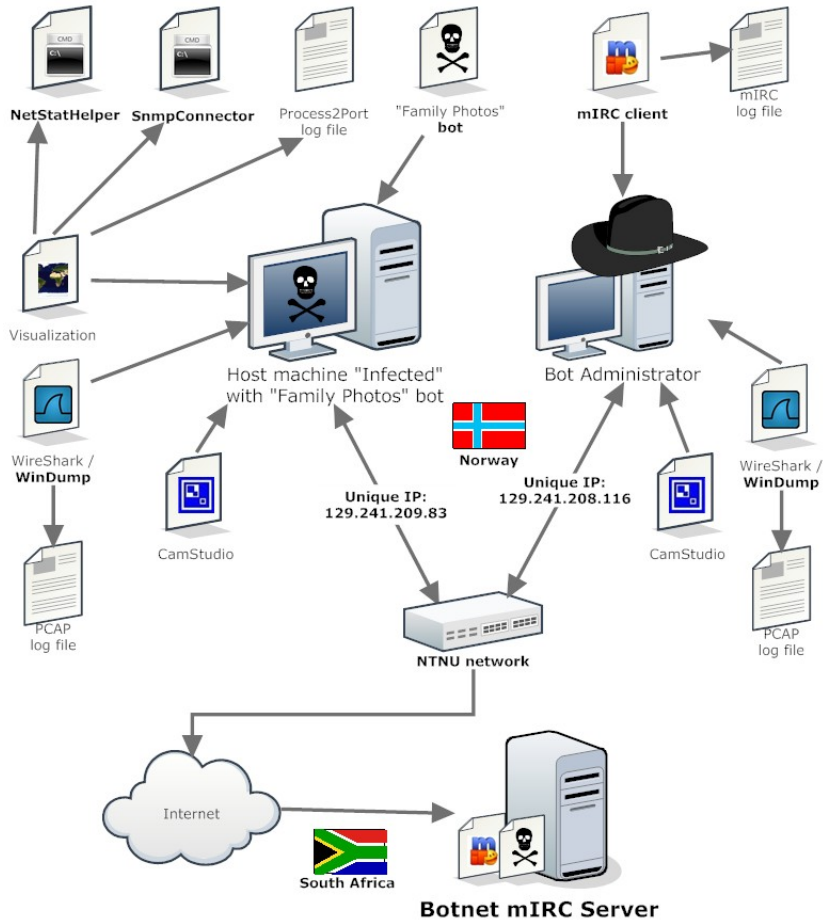


Figure 4.3: Extended experiment setup.

4. Wait for PC1 to complete to startup of the *ModifiedJavaBot.jar* bot.
5. Once the bot joins the channels, open a private conversation.
6. Send *.commands* to the bot and wait for response. This will ensure that the bot is responding correctly.
7. Once the bot has responded it is considered to be running correctly, send commands required by server administrators according to description in Section 4.2.3. This will ensure that the experiment is conducted in accordance with the preconditions set by the IRC server administrator.
8. Send *.getip* command to retrieve the bot's public IP for later analysis. This will ensure we are communicating with the correct computer and this forms the basis for a successful result.
9. Send *.exit* command to shut down the bot.
10. Close mIRC, CamStudio and WinDump.

### 4.3.2 Step by step guide for PC1

The instructions and tasks to be carried out for the bot infected computer is listed in sequential order. The purpose of these instructions is to make sure the instructions are carried out correctly and provide solid results to base the analysis on.

1. It is important to have the IP address of the computer for later analysis. This is obtained by running *ipconfig*, described in Section 2.6.5, in a command shell. This will be compared with the results from the *.getip* command performed by PC2 and results from the Visualization application.
2. Start WinDump [97] in a command shell with the following command: `WinDump.exe -n -e -i 1 -s 0 -w c:\experiment2_PC1_bot.pcap` For more information on WinDump, see Section 2.6.3. This will show the communication between the machines from the point of view of the infected machine, PC1.
3. Capture screenshots of *netstat* and *taskmanager/tasklist* to show behavior before bot is activated. This will be used for later comparison.

4. Start Visualization application and make sure *process2port.csv* is not present in the *saveCSV* folder, as described in Section B.3.2. This will ensure a clean base for the results.
5. Start CamStudio and focus the recording window on the area where the Visualization is running.
6. Start *ModifiedJavaBot.jar* as described in Section 4.2.4.
7. Capture screenshots of *netstat* and *taskmanager/tasklist* for comparison. This will show the bot's footprint on the system.
8. Allow for PC2 to carry out the bot commands.
9. Once PC2 has finished its list of instructions and tasks, close the bot if PC2 did not issue the *.exit* command.
10. Close Visualization, CamStudio and WinDump.

## 4.4 Results

Since there are multiple computers involved in this experiment, the results for PC1 and PC2 will be shown separately.

### 4.4.1 Results for PC1

To determine what IP PC1 has and to be able to compare it to the results from PC2, *netstat* was run. As the result shown in Figure 4.4, the IP address is: *129.241.209.83*

Before initiating the bot, *netstat* and *taskmanager* showed no connections to port 6667 and no unknown running processes, as shown in the first part of Figure 4.7 and Figure 4.5. Once the bot was started, a “Windows Security Alert” appeared, as seen in Figure 4.6. The alert asked the user to allow “Java Platform SE binary” to communicate to private and public networks. If a user would be confronted with this message, the assumption would be that the user would allow access to both “Private networks” and “Public networks”. Thus both options were checked off and “Allow access” was selected.

```

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : ed.ntnu.no
    IPv6 Address . . . . .           : 2001:700:300:1430:ac50:c132:bbe4:55fd
    Temporary IPv6 Address . . . . . : 2001:700:300:1430:215f:7c81:f3f3:8269
    Link-local IPv6 Address . . . . . : fe80::ac50:c132:bbe4:55fd%11
    IPv4 Address . . . . .           : 129.241.209.83
    Subnet Mask . . . . .            : 255.255.254.0
    Default Gateway . . . . .        : fe80::20c:cfff:fe32:4800%11
                                         129.241.208.1

```

Figure 4.4: ipconfig results for PC1.

wininit.exe	384	SYSTEM	00	92 K	Windows Start-Up Application
csrss.exe	392	SYSTEM	00	764 K	Client Server Runtime Process
winlogon.exe	432	SYSTEM	00	124 K	Windows Logon Application
services.exe	476	SYSTEM	00	1 608 K	Services and Controller app
lsass.exe	484	SYSTEM	00	1 304 K	Local Security Authority Process
lsm.exe	492	SYSTEM	00	420 K	Local Session Manager Service

Figure 4.5: Initial Taskmanager results for PC1.

The bot was now suspected to be running and so *netstat* and *taskmanager* was run again. The results can be seen in the second part of Figure 4.7 and Figure 4.8. A new connection had appeared and its PID was *464* and the foreign IP and port pair was: *196.210.236.37:6667*.

The *taskmanager* also showed a new process with PID *464* and image name of *javaw.exe*.

During the Visualization, several screen shots were captures with the “s” hot key. The activity that was captured are shown in Figure 4.9, Figure 4.10 and Figure 4.11.

Once PC2 was finished with its commands and the final *.exit* command was used, *netstat* and *taskmanager* results showed that the connection and process was gone. The Visualization, CamStudio and WinDump were then closed. The results from WinDump can be seen in Figure 4.12, and the *process2port.csv* log file that was created by the Visualization is shown below. Connections with no public IP have been omitted and the log format has been edited to improve readability. The format of the log file is explained in Section B.3.2:

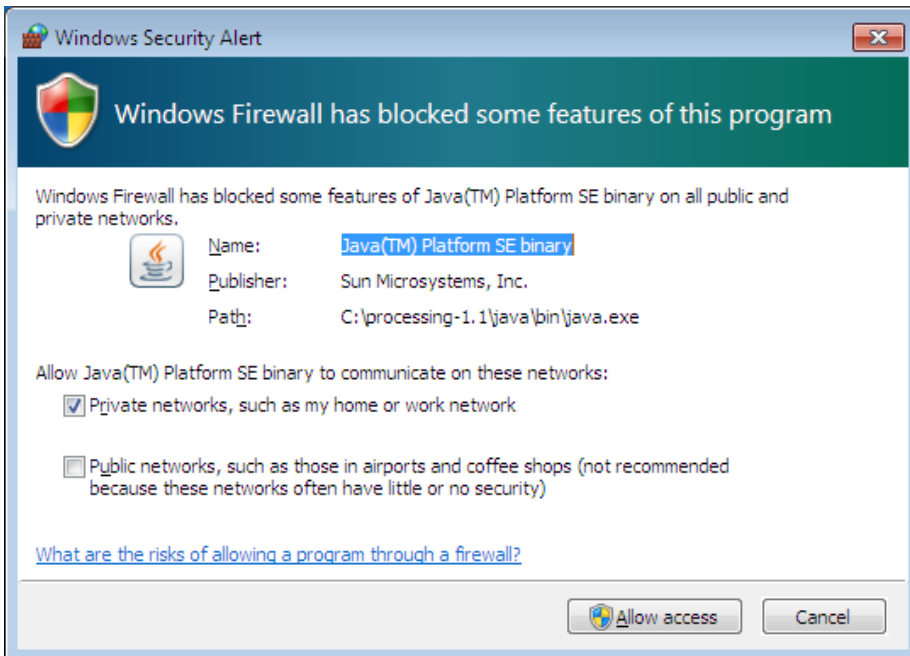


Figure 4.6: Windows Security Alert for Java Platform SE binary.

```

C:\Users\PC1>netstat -ano -p tcp
Active Connections
  Proto Local Address           Foreign Address         State       PID
  TCP   0.0.0.0:135              0.0.0.0:0               LISTENING  668
  TCP   0.0.0.0:445              0.0.0.0:0               LISTENING  4
  TCP   0.0.0.0:5357             0.0.0.0:0               LISTENING  4
  TCP   0.0.0.0:49152            0.0.0.0:0               LISTENING  384
  TCP   0.0.0.0:49153            0.0.0.0:0               LISTENING  724
  TCP   0.0.0.0:49154            0.0.0.0:0               LISTENING  884
  TCP   0.0.0.0:49155            0.0.0.0:0               LISTENING  476
  TCP   0.0.0.0:49156            0.0.0.0:0               LISTENING  484
  TCP   129.241.209.83:139      0.0.0.0:0               LISTENING  4
  TCP   129.241.209.83:49177    158.38.122.10:80        CLOSE_WAIT 1240

C:\Users\PC1>netstat -ano -p tcp
Active Connections
  Proto Local Address           Foreign Address         State       PID
  TCP   0.0.0.0:135              0.0.0.0:0               LISTENING  668
  TCP   0.0.0.0:445              0.0.0.0:0               LISTENING  4
  TCP   0.0.0.0:5357             0.0.0.0:0               LISTENING  4
  TCP   0.0.0.0:49152            0.0.0.0:0               LISTENING  384
  TCP   0.0.0.0:49153            0.0.0.0:0               LISTENING  724
  TCP   0.0.0.0:49154            0.0.0.0:0               LISTENING  884
  TCP   0.0.0.0:49155            0.0.0.0:0               LISTENING  476
  TCP   0.0.0.0:49156            0.0.0.0:0               LISTENING  484
  TCP   129.241.209.83:139      0.0.0.0:0               LISTENING  4
  TCP   129.241.209.83:49177    158.38.122.10:80        CLOSE_WAIT 1240
  TCP   129.241.209.83:49190    196.210.236.37:6667     ESTABLISHED 464
  TCP   129.241.209.83:49191    72.233.89.199:80        TIME_WAIT  0

```

Figure 4.7: netstat results for PC1 before and after *ModifiedJavaBot.jar* has been initiated.

wininit.exe	384	SYSTEM	00	92 K	Windows Start-Up Application
csrss.exe	392	SYSTEM	00	764 K	Client Server Runtime Process
winlogon.exe	432	SYSTEM	00	124 K	Windows Logon Application
javaw.exe	464	PC1	00	5 916 K	Java(TM) Platform SE binary
services.exe	476	SYSTEM	00	1 520 K	Services and Controller app
lsass.exe	484	SYSTEM	00	1 264 K	Local Security Authority Process
lsmd.exe	492	SYSTEM	00	408 K	Local Session Manager Service

Figure 4.8: Taskmanager results for PC1 after *ModifiedJavaBot.jar* has been initiated.



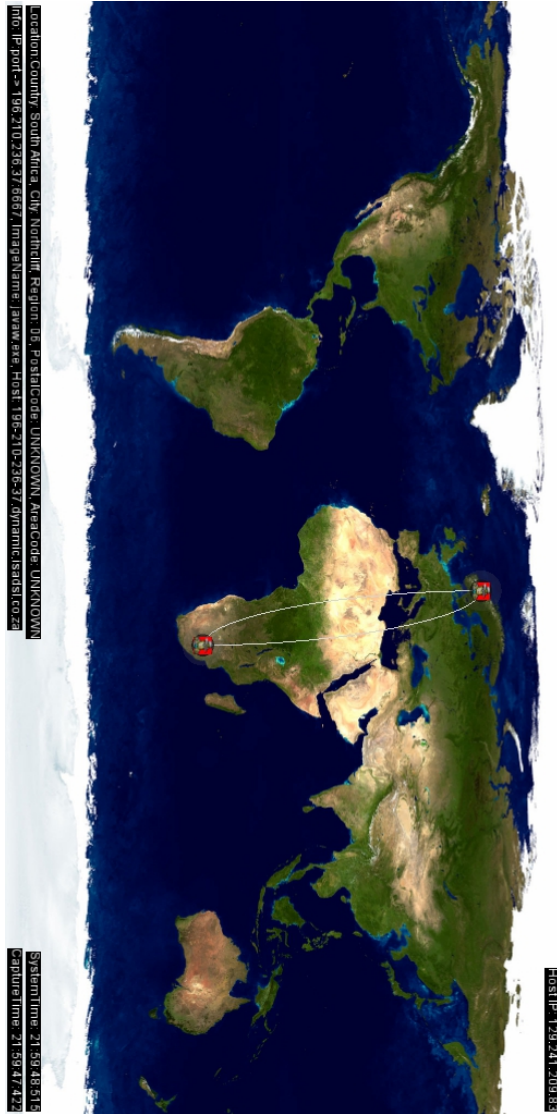


Figure 4.9: Visualization showing IRC communication between *ModifiedJavaBot.jar* and IRC server in South Africa.



Figure 4.10: Visualization zoomed in on IRC server in South Africa.

```
2010-05-31 21:59:12:134,129.241.209.83,49177,158.38.122.10,80,
CLOSE_WAIT,TCP_RFC,1240,jusched.exe,
C:\Program Files\Common Files\Java\Java Update\,
invalid,application,
```

```
2010-05-31 21:59:12:134,129.241.209.83,49190,196.210.236.37,6667,
ESTABLISHED,TCP_RFC,464,javaw.exe,
C:\Program Files\Java\jre6\bin\,invalid,application,
```

```
2010-05-31 21:59:12:134,129.241.209.83,49192,72.233.89.199,80,
TIME_WAIT,TCP_RFC,0,System Idle,
, ,operatingSystem,
```

#### 4.4.2 Results for PC2

To determine what IP PC2 has, *netstat* was run. As the result in Figure 4.13 shows, the IP address is: *129.241.208.116*

Once the mIRC client was started, a “Windows Security Alert” appeared, as

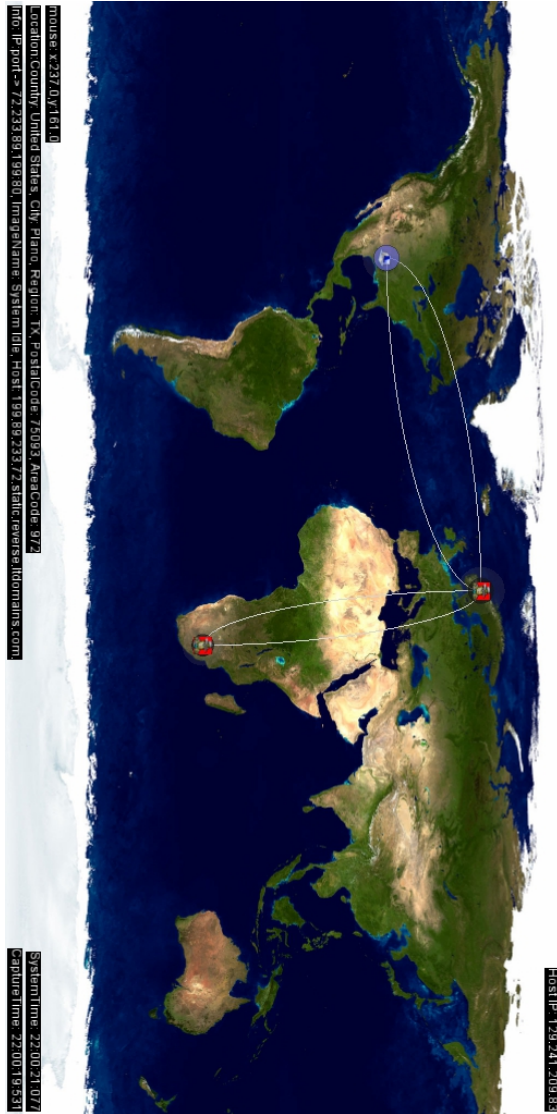


Figure 4.11: Visualization of connections made by *ModifiedJavaBot.jar* when PC2 issues the *.getip* command.

```

C:\>Windump.exe -n -e -i 1 -s 0 -w c:\experiment2_PC1_bot.pcap
Windump.exe: listening on \Device\NPF_{2C8B9320-8ECD-4EBE-8149-BDA9A65A392F}
23060 packets captured
23071 packets received by filter
0 packets dropped by kernel

```

Figure 4.12: Windump results for PC1.

```

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : ed.ntnu.no
    IPv4 Address. . . . . : 129.241.208.116
    Subnet Mask . . . . . : 255.255.254.0
    Default Gateway . . . . . : 129.241.208.1

```

Figure 4.13: ipconfig results PC2.

seen in Figure 4.14. The alert asked the user to allow “mIRC” to communicate to private and public networks. As with the “Windows Security Alert” encountered by PC1, the assumption would be that the user would allow access to both “Private networks” and “Public networks”. Thus both options was checked off and “Allow access” was selected.

Once the connection to the IRC server was established and the mIRC client was configured correctly, the channels were set up. At this point PC1 initiated the bot and it appeared in both IRC channels. A private conversation was set up with the bot, as shown in Figure 4.15, and the commands described in Section 4.3.1 was carried out. The response to the *.getip* command returned: *128.241.209.83*

Once the commands and *.exit* command had been executed, the bot left the channels with the message:

```

[22:02] * [TEST]-FIXED (~TEST-FIX@Webcomm-C3C7E409.item.ntnu.no)
Quit (Connection reset by peer)

```

The mIRC client, CamStudio and WinDump was then closed. The results from WinDump can be seen in Figure 4.16.

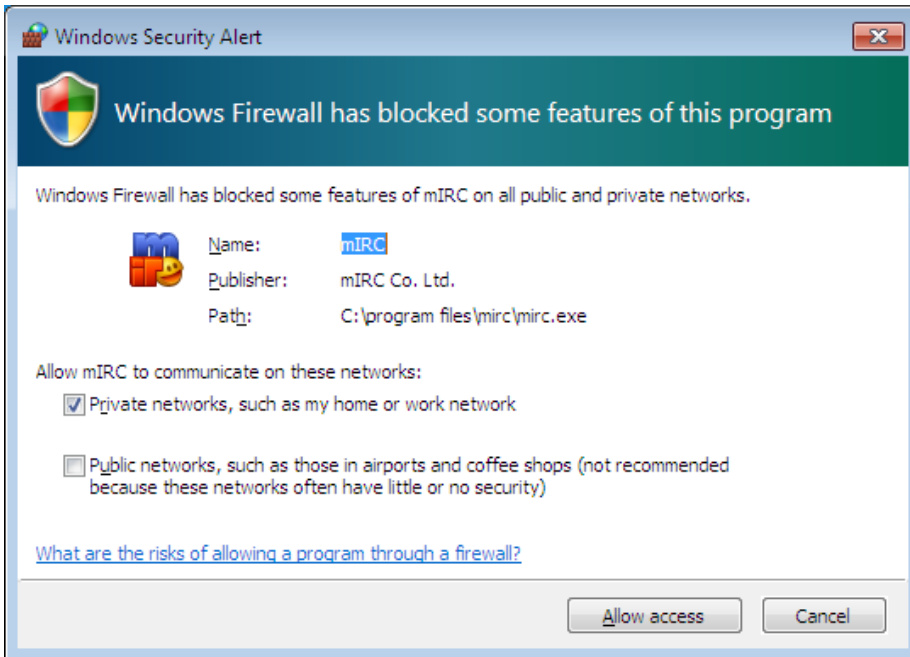


Figure 4.14: Windows Security Alert for mIRC.

```

[TEST]-FIXED (Webcomm, PC2Controller) (-TEST-FIX@Webcomm-C3C7E409.item.ntnu.no)
<PC2Controller> .commands
<[TEST]-FIXED> :: .commands (Shows this dialog (Neutralized)) ::
<[TEST]-FIXED> :: .exit (Shut down bot (Neutralized)) ::
<[TEST]-FIXED> :: .quit <message> (Leave IRC server (Neutralized)) ::
<[TEST]-FIXED> :: .join <channel> (Join channel (Neutralized)) ::
<[TEST]-FIXED> :: .part <channel> (Part channel (Neutralized)) ::
<[TEST]-FIXED> :: .raw <line> (Send raw line to IRC server (Neutralized)) ::
<[TEST]-FIXED> :: .getip <url [optional]> (Gets IP address, option to set URL to read from (Neutralized)) ::
<PC2Controller> .join #bots
<PC2Controller> .raw "MODE [TEST]-FIXED +B"
<PC2Controller> .getip
<[TEST]-FIXED> 129.241.209.83
<PC2Controller> .join #bots2
<PC2Controller> .getip
<[TEST]-FIXED> 129.241.209.83
<PC2Controller> .exit

```

Figure 4.15: IRC chat conversation between PC1 and PC2 with issued commands.

```
C:\>Windump.exe -n -e -i 1 -s 0 -w c:\experiment2_PC2_controller.pcap
Windump.exe: listening on \Device\NPF_{A072F88D-6C46-445A-973B-C53595FA6F3A}
29087 packets captured
29090 packets received by filter
0 packets dropped by kernel
```

Figure 4.16: Windump results for PC2.

## 4.5 Analysis

The analysis will first be carried out for PC1 and PC2 respectively. Then a final summary analysis will be given.

### 4.5.1 Analysis PC1

One thing that was not expected was the requirement of PC1 to scale down the image to 1024 since it could not cope with the current image size. The performance of the machine was also significantly lower than on the development machine and this also impacted the speed of the Visualization application. From this observation one can conclude that in order to run this type of application it is essential to have a high performance computer available.

Once the bot was up and running, one could not see the bot JAR file in the taskmanager and this would correspond to the claim, from Section 2.4, that Java based botnets are FUD. However, this is not entirely true as the *netstat* results showed the connection and its PID *464*. The results from *taskmanager* showed that PID *464* corresponded to *javaw.exe*. It is identical to *java.exe*, but there is an associated console window with *javaw.exe*. Since the file that is running is shown as a different process, this will make it more difficult for a user to see what is actually running in the background. The port 6667 used by the remote IP could raise suspicion as it is the default port for IRC, but most users will not look at *netstat* data regularly. With a simple change in the server setup this port can be changed and since many botnets use UnrealIRCd [86] server, this is trivial and does not prove definitively the presence of a botnet.

In the Visualization *javaw.exe* was “blacklisted” in the way that the image icon for this process was changed, but the connection itself was still detected.

The image icon was meant as a confirmation if the experiment was successful in mapping the process. The initial connection to the IRC server is shown in Figure 4.9. The Visualization application shows that the server is located in Northcliff, South Africa and it's IP and port pair is *196.210.236.37:6667*. The *netstat* results in Section 4.4.1 confirm that this is the connection of the bot. The server location also confirms that this is the IRC server described in Section 4.2.3. In Figure 4.10 one can see a close-up of the IRC server.

Once the bot administrator, issues the *.getip* command, the bot contacts *whatismyip* [101] to retrieve the public IP. The result of this is shown in Figure 4.11.

An analysis of the packet capture performed by WinDump confirms the connection. The packets were filtered out with the following rule:

```
(ip.src == 129.241.209.83 || ip.dst == 129.241.209.83) &&
(tcp || http || dns)
```

The *.getip* command from the bot controller, PC2, is shown in Figure 4.17 and the returned response is shown in Figure 4.18.

98 22:00:05.584688	196.210.236.37	129.241.209.83	IRC	Response
<ul style="list-style-type: none"> <li>▣ Frame 98 (140 bytes on wire, 140 bytes captured)</li> <li>▣ Ethernet II, Src: Cisco_32:48:00 (00:0c:cf:32:48:00), Dst: DellPcba_f9:42:bc (00:0d:56:f9:42:bc)</li> <li>▣ Internet Protocol, Src: 196.210.236.37 (196.210.236.37), Dst: 129.241.209.83 (129.241.209.83)</li> <li>▣ Transmission Control Protocol, Src Port: ircu (6667), Dst Port: 49190 (49190), Seq: 9838, Ack: 793, Len: 86</li> <li>▣ Internet Relay chat <ul style="list-style-type: none"> <li>Response: :PC2Controller!PC2Control@webcomm-482F214D.item.ntnu.no PRIVMSG [TEST]-FIXED :.getip</li> </ul> </li> </ul>				

Figure 4.17: PC1 Wireshark request with the *.getip* command.

105 22:00:05.957026	129.241.209.83	196.210.236.37	IRC	Request
<ul style="list-style-type: none"> <li>▣ Frame 105 (92 bytes on wire, 92 bytes captured)</li> <li>▣ Ethernet II, Src: DellPcba_f9:42:bc (00:0d:56:f9:42:bc), Dst: Cisco_32:48:00 (00:0c:cf:32:48:00)</li> <li>▣ Internet Protocol, Src: 129.241.209.83 (129.241.209.83), Dst: 196.210.236.37 (196.210.236.37)</li> <li>▣ Transmission Control Protocol, Src Port: 49190 (49190), Dst Port: ircu (6667), Seq: 793, Ack: 9924, Len: 38</li> <li>▣ Internet Relay chat <ul style="list-style-type: none"> <li>Request: PRIVMSG PC2Controller :129.241.209.83</li> </ul> </li> </ul>				

Figure 4.18: PC1 Wireshark response to the *.getip* command.

The *process2port.csv* log file also shows the connection made by *javaw.exe*. With this connection entry logged, we have documented the connection with

a date stamp in a way that would not otherwise be possible with *netstat* and *taskmanager*.

## 4.5.2 Analysis PC2

An analysis of the packet capture performed by WinDump again confirms the connection and the results of the Wireshark captures from PC1 were filtered out with the following rule:

```
(ip.src == 129.241.208.116 || ip.dst == 129.241.208.116) &&
(tcp || http || dns)
```

The *.getip* command sent from the *mIRC* client was captured by Wireshark and the IP that was returned corresponds to the public IP address of PC1, as explained in previous sections. The request and response from the PC2 packet capture is shown in Figure 4.19 and the returned response is shown in Figure 4.20.

15342	21:58:17.890185	129.241.208.116	196.210.236.37	IRC	Request
[-] Frame 15342 (83 bytes on wire, 83 bytes captured)					
[-] Ethernet II, Src: DellPcba_ca:12:9e (00:0d:56:ca:12:9e), Dst: Cisco_32:48:00 (00:0c:cf:32:48:00)					
[-] Internet Protocol, Src: 129.241.208.116 (129.241.208.116), Dst: 196.210.236.37 (196.210.236.37)					
[-] Transmission Control Protocol, Src Port: 49182 (49182), Dst Port: ircu (6667), Seq: 697, Ack: 10556, Len: 29					
[-] Internet Relay Chat					
Request: PRIVMSG [TEST]-FIXED :.getip					

Figure 4.19: PC2 Wireshark request with the *.getip* command.

15448	21:58:20.559678	196.210.236.37	129.241.208.116	IRC	Response
[-] Frame 15448 (147 bytes on wire, 147 bytes captured)					
[-] Ethernet II, Src: Cisco_32:48:00 (00:0c:cf:32:48:00), Dst: DellPcba_ca:12:9e (00:0d:56:ca:12:9e)					
[-] Internet Protocol, Src: 196.210.236.37 (196.210.236.37), Dst: 129.241.208.116 (129.241.208.116)					
[-] Transmission Control Protocol, Src Port: ircu (6667), Dst Port: 49182 (49182), Seq: 10556, Ack: 726, Len: 93					
[-] Internet Relay Chat					
Response: :[TEST]-FIXED!~TEST-FIX@webcomm-C3C7E409.item.ntnu.no PRIVMSG PC2controller :129.241.209.83					

Figure 4.20: PC2 Wireshark response to the *.getip* command.

The *mIRC* logs from PC2 also confirm these findings, but there are some discrepancies. The timestamps show a slightly different time than files from PC1.



This is because the clocks on the computers were not synchronized properly, but this does not affect the results as the time plays a marginal factor in the visualization and detection.

### 4.5.3 Summary analysis

The results from both computers show that the botnet seen by PC2 corresponded to the IP of PC1. The *process2port.csv* log file, Wireshark results, *netstat* results and Visualization screen shots all confirm the presence of the bot.

The botnet IRC server was also detected by the infected host as shown in the logs from *netstat*, *taskmanager* and Visualization as well as in the Visualization application itself. Since no specific adjustments for Java based botnet detection and visualization was used in the environment, the techniques used are general and this point towards a wider use of the Process2Port library and the Visualization.

This experiment shows that it is possible to detect and visualize a botnet with the use of *process-to-port* mapping and this has been shown in the images captured from the Visualization application. The claim that Java based botnets is Fully UnDetectable (FUD) does not hold true according to the results of this experiment.



# Chapter 5

## Discussion

“If it bleeds, we can kill it.”—Arnold Schwarzenegger, Actor and 38th Governor of California. From the movie Predator (1987)

As stated throughout this thesis, botnets are one of the biggest threats one can face in cyberspace today. They have the ability to launch attacks, perform espionage and hand over control of the computer. I would argue that the difficulty to detect their presence is the main reason they can cause such damage. As the quote the start of this chapter tries to convey, when you find the presence of the botnet it is much easier to fight it. I believe that a botnet’s greatest weakness is its need for communication to its C2 rendezvous point. The inherent structure makes it dependent on communication with its botnet administrator. This communication that is visualized in the experiment clearly shows the communications made by the bot residing on the host machine, including external connections made by the bot.

The Visualization application and Process2Port library used in the visualization of botnets is based on white listing known applications and displaying these with an image icon. In the experiment this was somewhat reversed by changing the image icon associated with *javaw.exe* to represent the presence of a botnet. The bot was detected with the Visualization application, but it was not automatically classified as a botnet. I believe there is a big difference with regards to detection and classification. A user viewing the Visualization

and observing the botnet is tasked with classifying the connections as those of a botnet. I believe this method of detection is also in agreement with “A Host-Based Approach to BotNet Investigation?” [36]. This thesis has focused on visualizing an IRC-based botnet, but the detection scheme employed can easily be changed or extended to include related works on botnet detection from Section 2.3.

The Visualization application and Process2Port library used in the experiment was used for visualization and detection of a botnet from the perspective of a host computer. However, the technologies and design choices made throughout the thesis have a strong focus on showing that the results can also be used in a distributed context. SNMP was specifically chosen because of its built in support for deployment in a distributed environment. What was not expected was that some of the key OIDs turned out to not function properly. This is most likely because they were too new to have been implemented. This made it necessary to use *netstat* to be able to map running processes and connections together, which limits the ability to focus on distributed technologies as one would need to install custom software on each computer involved. Once the missing OIDs are implemented the principles used in this thesis can be used in combination with a central authority that analysis connections on a network, thus creating a distributed system for visualizing a larger network.

By using SNMP agents that are already bundled with Windows, I believe this can provide a valuable tool for system administrators. If one also combines packet captures from a central authority with what is seen by the host computers through SNMP, rootkits employed to hide a bot from the host system may still be detected by the discrepancy between results from the host computer and the central authority. This may provide a significant increase in detection rate of botnets running on a network. Using visualization for representing data also reduces the threshold for user to understand all aspects of the system. The threshold is reduced since the most technical parts are abstracted away and replaced with informative nodes.

As botnets use well established channels of communication that is normally utilized by legitimate application, I believe that the question of automatic detection and classification of the presence of a botnet boils down to mapping human behavior in these communication channels. This does require a much deeper analysis of the data and one will no longer be limited to IP and port as packet analysis may require analysis of the payload. The issue of privacy for the users as well as the legal questions that might arise must then be taken

into consideration. By solely relying on IP and port, which is available to all intermediate points in the communication and can be regarded as somewhat public, the techniques presented in this thesis could provide a win-win situation.

I argue that botnets and bot applications are basically the same as any other legitimate application. The difference is that bots use the existing communication functionality, originally associated with proprietary applications, to communicate with C2 rendezvous point or carry out malicious tasks. As shown from the analysis of the bots in Section 2.7, this also involves using existing functionality for constructing packets in order to create attack capabilities. Botnets are a weapon and they can and have been used for that purpose. As the quote from Appendix B states; any tool is a weapon if you hold it right. This is particularly true for botnets as they use existing functionality for uses not intended by the original authors.

The majority of related works done on botnet detection seem to focus on creating advanced mathematical metrics and packet analysis to detect the presence of a botnet. The results of the experiment show that this may not be necessary in order to detect the presence of a bot running on the host computer. Although the *process-to-port* mapping technique requires more user interaction, I believe that it could provide an alternative to other forms of detection or function as a supplemented to existing administration and monitoring tools.

As mentioned in Section 3.1, humans intuitively create mental maps in order to navigate in the world. Based on research by Lynch on how humans navigate, from Section 3.1, the elements Lynch describes also fit quite well with the elements in the Visualization. By relying on intuitive knowledge from the user in combination with the Visualization application, I believe that one would intuitively try to make a mental map and with prolonged usage one could spot patterns in the communication on the network. This could presumably help finding new ways of viewing a network, finding discrepancies and in conjunction with other tools can help viewing the data available in a new way.

The ability to color-code and change animation of events shown in the Visualization application, can also be further utilized if combined with other systems that produces alarms when faults are detected, such as Intrusion Detection System (IDS)/Intrusion Prevention System (IPS) systems. Instead of classifying as “alarm” or “no alarm”, the ability to change the attributes of the elements available makes it possible to classify the severity of a particular event and produce the results to the user. This could help reduce the false alarm rates of

exciting systems as well as presenting the data in a more visually stimulating way than looking through log files.

Results shows that the principles with regard to *process-to-port* mapping can be very useful in detecting botnets, but an unanticipated observation from the Visualization and *process-to-port* mapping was that one get access to network communication in a new way. The main motivation behind the thesis was “Visualizing Spatial and Temporal Dynamics of a Class of IRC-Based Botnets”, but the resulting Visualization application shows more than just botnets. It allows for visualizing all communicating applications and connections to and from the host computer. This opens up for use in other areas than just botnets such as investigating communications made by other applications as well, making this a tool that can be used for more general purposes.

During the development of the Visualization application, TCP flags were added to the debug console of Processing. One of the reasons for this was to be able to see if the packets were part of an initial connection, but an observation showed that there were a disproportionate number of SYN flags. At first I believed what I was seeing was a SYN attack on a computer on the network, but later found out that due to the large number of broadcast packets on the network this assumption was not true. Even though the observation turned out to be false, it still points out the usefulness of displaying TCP flags to the user. If the host machine infected with a bot was part of a botnet attack on an external server, this would presumably be shown in the flags.

The choice to focus on the bot connection itself and not on the volume of traffic, proved to be a good choice for this context, as was the choice to group connections into a single node and assigning an image icon corresponding to the application responsible for the connection. The network the Visualization application was tested on has a high traffic volume and the test applications also produced many distinct connections. It is easy to see what connections correspond to which application and the differences in the image icons helped distinguish between processes. Instead of permanently changing the node qualities based on the traffic volume, each node would “activate” each time a packet was received from a specific IP. This proved very useful as it clearly showed the network activity at that particular moment in time. It does not seem to be important to take the actual volume of traffic into account as the activity in the Visualization provided a clear picture of what was going on. Some of the test applications used P2P technology and produced numerous distinct nodes. This made the Visualization quite cluttered, but implementing a functionality to filter out

specific applications or connections will remove this issue. Using geography as an integral part of the Visualization proved very rewarding. It made it easy put the information from the packet capture into context and made it possible to see the data from a new perspective. The location and time of an observation also formed new questions that might not have arisen without the Visualization.

In the Visualization application labels are used to display added information. This includes specific information for a particular connection if a particular node is selected. Location, IP, port and hostname was primarily the data that was shown. After the node has been selected and the information is presented to the user, but it is difficult to see which node the information is relevant for. This is an area for improvement and integrating the node and its information in a more informative way would greatly improve the usability of the Visualization application.





# Chapter 6

## Conclusion

“Know thy self, know thy enemy. A thousand battles, a thousand victories.”—Sun Tzu

The results from the experiments show that it is possible to detect and visualize the presence of a bot on a host computer. Although the Visualization and Process2Port library used were able to detect and visualize the botnet, it did not automatically classify the connection as malicious. White listing was employed to give the communicating applications the correct image icons and the bot used in the experiment was classified beforehand. The metrics involved to create this system is based on host based metrics, IP and port. The technique of *process-to-port* mapping was able to show connections made by the bot, but also all other communicating applications running on the host system. This seems to point to a wider use of the Visualization application for other purposes than just botnet detection and visualization.

The Process2Port library was successful in mapping each connection an application was responsible for. It was meant to run on a host system, but the technologies implemented also shows that it can be used to create a distributed system based on the same principles as for the single host. To be able to further develop a distributed system without any additional software installed on the involved computers, new SNMP OIDs need to be supported.

The Visualization application successfully produced a good overview of the

traffic flowing to and from the host machine. Differences with regards to the communicating applications are displayed in a distinct and clear way. Using geography as an integral part of the Visualization proved to be a good choice. It formed new and interesting questions about the observations and presented the information from a new perspective.

The bot on the host machine was clearly shown as well as the connections it was responsible for. Results from the experiment and general testing of the Visualization proved it to be a good tool for general network visualization even though it was designed for botnet detection and visualization. This would indicate that it might contribute with something new to the field of visualization.

Botnets are a weapon and the threat posed should not be underestimated and the damage these networks can inflict on the host system and to other networks can be severe. The successful detection of this malware is of prime importance both for home users, businesses and countries.

# Chapter 7

## Future work

“The best way to predict the future is to invent it.”—Alan Kay

During the development of the Process2Port library and Visualization application a number of possible extensions were discovered, but due to time constraints and to limit the scope of the thesis they were omitted. Some of these possible extensions will be explored in this section.

The Process2port library uses host based metrics in the form of SNMP and *netstat* to create the *process-to-port* mapping of the connections to and from the host machine. Added functionality with regards to mapping applications could be extended with the use of *Java Virtual Machine Process Status Tool*. This can among other features list the processes that are running on the JVM and can further help to distinguish legitimate Java processes from the malicious ones as shown by the Java based botnet. Adding functionality to map connection hops, with the use of *traceroute/tracert*, could provide additional information about which communication trunks the information is likely to pass through. Extracting the favicons from visited websites would help give more information about what sites are accessed if one is monitoring a network.

A mapping scheme to parse logs from firewalls, IDS/IPS and other security systems would add to the intelligence of the nodes as alarms and suspicious findings could be color coded in the Visualization. Collection of host based data should ideally be done centrally and a mechanism for sending host metrics

to a central server for processing would also make the Visualization provide a bigger picture of the network as well as reducing the performance impact on each computer. The ability to export the data to other formats such as for Google Earth would make the library useful for other visualization applications as well.

The Visualization application could be improved by creating a more illustrative result by including information about night or day in a particular location. I attempted to implement this in the current Visualization, but the advanced mathematics of calculating the position of the sun added significant complexity and decreased the performance of the application, so it was omitted. Instead of using a global map of the world, it might be more useful for system administrators to use geolocation for a smaller geographical area such as an university campus to map the usage of different applications or occurrences of malware.

Ability to play, pause, forward and rewind the Visualization would make it easier to step back and investigate suspicious activity without having to monitor in real time. Being able to load and visualize previously captured data, would enable the application to function as a demonstration tool as well as real time monitoring. Since there may be significant traffic on larger networks, the ability to filter out certain applications, such as Internet browsers, would help make unusual behavior stand out and reduce clutter. The image icons are currently manually mapped to the application name, but extracting the image icon directly from the observed executable would reduce the user's involvement in setting up the Visualization application. This would also improve the usability and allow other applications to use the same concept of mapping image icons to applications much more easily.

# Bibliography

- [1] Akamai. The state of the internet 3rd quarter 2009. Report, 2009. [http://www.akamai.com/dl/whitepapers/Akamai\\_State\\_Internet\\_Q3\\_2009.pdf](http://www.akamai.com/dl/whitepapers/Akamai_State_Internet_Q3_2009.pdf).
- [2] Internet Assigned Numbers Authority. <http://www.iana.org/assignments/port-numbers>. <http://www.iana.org/assignments/port-numbers>.
- [3] Thomas Barkowsky. *Mental Representation and Processing of Geographic Knowledge*, volume 2541 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002. <http://www.springerlink.com/content/01j11k5k11jg/>.
- [4] Richard Bejtlich. *Extrusion Detection*. Addison-Wesley, 1st edition, 2005.
- [5] Casey Reas Ben Fry. Processing. <http://processing.org/>.
- [6] James R. Binkley and Suresh Singh. An algorithm for anomaly-based botnet detection. Paper, 2006. <http://web.cecs.pdx.edu/~jrb/jrb.papers/sruti06/sruti06.pdf>.
- [7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 10 2007.
- [8] Bill Blunden. *The Rootkit Arsenal*. Wordware Publishing, Inc, 1st edition, 2009.
- [9] John Canavan. The evolution of malicious irc bots. Paper, 2005. [http://www.symantec.com/avcenter/reference/the\\_evolution.of.malicious.irc.bots.pdf](http://www.symantec.com/avcenter/reference/the_evolution.of.malicious.irc.bots.pdf).

- [10] Joe Casad. *Sams Teach Yourself TCP/IP in 24 Hours*. Sams, 4th edition, 2008. <http://www.amazon.com/Sams-Teach-Yourself-TCP-Hours/dp/0672329964>.
- [11] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). RFC 1157 (Standard), May 1990. <http://www.ietf.org/rfc/rfc1157.txt>.
- [12] Common Event Expression (CEE). Common event expression white paper. Paper, 2007. <http://cee.mitre.org/docs/Common-Event-Expression-White-Paper.pdf>.
- [13] Patrick Charles. Jpcap - network packet capture facility for java. <http://jpcap.sourceforge.net/>.
- [14] CamStudio community. Camstudio - free streaming video software. <http://camstudio.org/>.
- [15] David Harley Craig A. Schiller, Jim Binkley. *Botnets - The killer web app*. Syngress, 1st edition, 2007.
- [16] Hauke Dämpfling. Gnutella web caching system. <http://gnucleus.sourceforge.net/gwebcache/>.
- [17] Datatilsynet. Datalagringsdirektivet, 2008. [http://www.datatilsynet.no/templates/article\\_\\_\\_2156.aspx](http://www.datatilsynet.no/templates/article___2156.aspx).
- [18] Joshua Davis. Hackers take down the most wired country in europe. Article, 2007. [http://www.wired.com/politics/security/magazine/15-09/ff\\_estonia?currentPage=all](http://www.wired.com/politics/security/magazine/15-09/ff_estonia?currentPage=all).
- [19] Design and analysis of communication systems Group. Simpleweb, 2010. <http://www.simpleweb.org/>.
- [20] Jack Dorsey, Evan Williams, and Biz Stone. Twitter. <http://twitter.com/about>.
- [21] Steven Dutch. The universal transverse mercator system, 2010. <http://www.uwgb.edu/DutchS/FieldMethods/UTMSystem.htm>.
- [22] DynDNS. Dyndns. <http://www.dyndns.com>.
- [23] Alexander Galloway. Carnivore - processing library. <http://r-s-g.org/carnivore/>.

- [24] Frederic Giroire, Jaideep Chandrashekar, Nina Taft, Eve Schooler, and Dina Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Recent Advances in Intrusion Detection*, volume 5758 of *Lecture Notes in Computer Science*, pages 326–345. Springer Berlin / Heidelberg, 2009. <http://www.springerlink.com/content/d537672q64130684>.
- [25] Andreas Groscurth, Thomas Cornet, Richard Welteroth, Steve Bromley, and Francesc Rosés. Fat jar eclipse plug-in. <http://fjep.sourceforge.net/>.
- [26] Christopher W. Hanna. Using snort to detect rogue irc bot programs. Technical report, 2004. [http://www.giac.org/certified\\_professionals/practicals/gsec/4095.php](http://www.giac.org/certified_professionals/practicals/gsec/4095.php).
- [27] Andreas Hegna, Arne Oslebo, and Morten Knutsen. Meeting with uninett, 02 2010. <http://www.uninett.no/>.
- [28] Andreas Hegna and Stig Henning Verpe. Meeting with ntnu it, 02 2010. <http://www.ntnu.no/adm/it>.
- [29] Erik Hjelmvik. Networkminer. <http://sourceforge.net/projects/networkminer/>.
- [30] Reid Hoffman. LinkedIn. <http://press.linkedin.com/about>.
- [31] IDC. International data corporation (idc) - press release. Press Release, 2009. <http://www.idc.com/getdoc.jsp?containerId=prUS22008509>.
- [32] IP2Location. Ip2location - bringing geography to the internet. <http://www.ip2location.com/>.
- [33] Jolo, prysm, and RuyDuck of EFnet IRC#help(channel). The new irc channel operator’s guide. <http://irchelp.org/irchelp/irctutorial.html>.
- [34] Chris Brown Ken Baylor. Killing botnets - a view from the trenches. Paper, 2006. [http://www.mcafee.com/us/local\\_content/white\\_papers/wp\\_botnet.pdf](http://www.mcafee.com/us/local_content/white_papers/wp_botnet.pdf).
- [35] Alexey Klimkin. edonkey protocol. <http://kent.dl.sourceforge.net/pdonkey/eDonkey-protocol-0.6.2.html>.

- [36] Frank Y. W. Law, K. P. Chow, Pierre K. Y. Lai, and Hayson K. S. Tse. A host-based approach to BotNet investigation? In *Digital Forensics and Cyber Crime*, volume 31 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 161–170. Springer Berlin Heidelberg, 2010. <http://www.springerlink.com/content/x28561g15882k83r/>.
- [37] Robert Lee and Sheau-Dong Lang. Locality-based server profiling for intrusion detection. In *Intelligence and Security Informatics*, volume 5075 of *Lecture Notes in Computer Science*, pages 205–216. Springer Berlin / Heidelberg, 2010.
- [38] Kevin Lynch. *The Image of the City*. The MIT Press, 1960. <http://www.amazon.com/Image-City-Kevin-Lynch/dp/0262620014>.
- [39] Khaled Mardam-Bey. mirc - internet relay chat client. <http://www.mirc.com/>.
- [40] Raffael Marty. Afterglow. <http://afterglow.sourceforge.net/>.
- [41] Raffael Marty. *Applied Security Visualization*. Addison-Wesley, 1st edition, 2009.
- [42] Maxmind. Maxmind - geolite city. <http://www.maxmind.com/app/geolitecity>.
- [43] Claudio Mazzariello and Carlo Sansone. Anomaly-based detection of IRC botnets by means of one-class support vector classifiers. In *Image Analysis and Processing ICIAP 2009*, volume 5716 of *Lecture Notes in Computer Science*, pages 883–892. Springer Berlin / Heidelberg, 2009.
- [44] McAfee. McAfee, inc. reports botnets threaten national infrastructure and security. Article, 2006. [http://www.mcafee.com/us/about/press/corporate/2006/20061024\\_155025\\_f.html](http://www.mcafee.com/us/about/press/corporate/2006/20061024_155025_f.html).
- [45] McAfee. McAfee q2 threats report reveals spam, botnets at an all time high. Article, 2009. [http://newsroom.mcafee.com/article\\_display.cfm?article\\_id=3545](http://newsroom.mcafee.com/article_display.cfm?article_id=3545).
- [46] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based internets. RFC 1156 (Standard), May 1990. <http://www.ietf.org/rfc/rfc1156.txt>.



- [47] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. RFC 1213 (Standard), March 1991. <http://www.ietf.org/rfc/rfc1213.txt>.
- [48] Trend Micro. The internet infestation, how bad is it really? Article, 2009. <http://blog.trendmicro.com/the-internet-infestation-how-bad-is-it-really/>.
- [49] Microsoft. Ipconfig. <http://technet.microsoft.com/en-us/library/bb490921.aspx>.
- [50] Microsoft. netstat. <http://technet.microsoft.com/en-us/library/bb490947.aspx>.
- [51] Microsoft. Nslookup. <http://technet.microsoft.com/en-us/library/bb490950.aspx>.
- [52] Microsoft. Ping. <http://technet.microsoft.com/en-us/library/bb490968.aspx>.
- [53] Microsoft. tasklist. <http://technet.microsoft.com/en-us/library/bb491010.aspx>.
- [54] Microsoft. Tracert. <http://technet.microsoft.com/en-us/library/bb491018.aspx>.
- [55] Microsoft. Windows management instrumentation command-line (wmic). <http://technet.microsoft.com/en-us/library/bb742610.aspx>.
- [56] Microsoft. About snmp, 2009. [http://msdn.microsoft.com/en-us/library/aa377941\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa377941(VS.85).aspx).
- [57] Microsoft. System files for snmp, 2010. [http://msdn.microsoft.com/en-us/library/aa379149\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa379149(v=VS.85).aspx).
- [58] Aaron Koblin Mike Chang. Textured sphere. Source code. <http://processing.org/learning/3d/texturedsphere.html>.
- [59] Mike Mintz and Andrew Sayers. Msn messenger protocol. <http://www.hypothetic.org/docs/msn/>.
- [60] Nasa. Blue marble. <http://earthobservatory.nasa.gov/Features/BlueMarble/>.

- [61] Arbor networks. Summary report global botnets. Report, 2010. <http://atlas.arbor.net/summary/botnets>.
- [62] Aaron Hackworth Nicholas Ianelli. Botnets as a vehicle for online crime. Article, 2005. <http://www.cert.org/archive/pdf/Botnets.pdf>.
- [63] No-IP. No-ip.com. <http://www.no-ip.com>.
- [64] JD (Justis og politidepartementet). Almindelig borgerlig straffelov (straffeloven)(Å§122), 2009. <http://www.lovdatab.no/cgi-wift/ldles?doc=/all/t1-19020522-010-015.html&122>.
- [65] JD (Justis og politidepartementet). Almindelig borgerlig straffelov (straffeloven)(Å§145), 2009. <http://www.lovdatab.no/cgi-wift/ldles?doc=/all/t1-19020522-010-017.html&145>.
- [66] J. Oikarinen and D. Reed. Rfc1459 - internet relay chat protocol. RFC, 1993. <http://irchelp.org/irchelp/text/rfc1459.txt>.
- [67] Gunter Ollmann. Botnet communication topologies. Paper, 2009. [http://www.damballa.com/downloads/r\\_pubs/WP%20Botnet%20Communications%20Primer%20\(2009-06-04\).pdf](http://www.damballa.com/downloads/r_pubs/WP%20Botnet%20Communications%20Primer%20(2009-06-04).pdf).
- [68] Younghee Park and Douglas S. Reeves. Identification of bot commands by run-time execution monitoring. In *Proceedings of the 2009 Annual Computer Security Applications Conference, ACSAC '09*, pages 321–330, Washington, DC, USA, 2009. IEEE Computer Society.
- [69] The Swedish Parliament. Accounting proposal points 2007/08, 2007. <http://www.riksdagen.se/webbnav/?nid=3154&rm=2007/08&bet=F%C3%B6U15>.
- [70] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. <http://www.ietf.org/rfc/rfc791.txt>.
- [71] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. <http://www.ietf.org/rfc/rfc793.txt>.
- [72] Open Processing. Open processing. <http://www.openprocessing.org/>.
- [73] Tor project. Tor - anonymity online. <http://www.torproject.org/>.
- [74] Tor project. Tor project - ip-to-country.csv. <http://ip-to-country.webhosting.info/downloads/ip-to-country.csv.zip>.

- [75] R. Raghunathan. Management Information Base for the Transmission Control Protocol (TCP). RFC 4022 (Standard), March 2005. <http://www.ietf.org/rfc/rfc4022.txt>.
- [76] Mark Russinovich, Bryce Cogswell, and Microsoft. Sysinternals. <http://technet.microsoft.com/en-us/sysinternals/default.aspx>.
- [77] Philippe Simonet. Getif network tool. <http://www.wtcs.org/snmp4tpc/getif.htm>.
- [78] Hispasec Sistemas. Virustotal - free online virus and malware scan. <http://www.virustotal.com/>.
- [79] Bill Soudan and Denis V. Dmitrienko. Icqlib: The icq library. <http://sourceforge.net/projects/icqlib/>.
- [80] Open source community. Net-snmp. <http://www.net-snmp.org/>.
- [81] Inc Sourcefire. Snort. <http://www.snort.org/>.
- [82] Joe Stewart. Sinit p2p trojan analysis. Article, 2003. <http://www.secureworks.com/research/threats/sinit/>.
- [83] Joe Stewart. Phatbot trojan analysis. Article, 2004. <http://www.secureworks.com/research/threats/phatbot/>.
- [84] Reto Stöckli, Eric Vermote, Nazmi Saleous, Robert Simmon, and David Herring. The blue marble next generation - a true color earth dataset including seasonal dynamics from modis. Paper, 2005. <http://earthobservatory.nasa.gov/Features/BlueMarble/bmng.pdf>.
- [85] W. Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. In *Botnet Detection*, volume 36 of *Advances in Information Security*, pages 1–24. Springer US, 2008.
- [86] Nick: Stskeeps. Unrealircd - open source irc daemon. <http://www.unrealircd.com/>.
- [87] Mani Subramanian. *Network Management: An Introduction to Principles and Practice*. Addison Wesley, 1st edition, 1999. <http://www.amazon.co.uk/Network-Management-Introduction-Principles-Practice/dp/0201357429>.

- [88] Tcpdump team. Tcpdump / libpcap. <http://www.tcpdump.org/>.
- [89] Tcpdump team. Tcpdump / libpcap manual. [http://www.tcpdump.org/tcpdump\\_man.html](http://www.tcpdump.org/tcpdump_man.html).
- [90] WASTE Development Team. Waste protocol. <http://waste.sourceforge.net/>.
- [91] Wireshark team. dumpcap. <http://www.wireshark.org/docs/man-pages/dumpcap.html>.
- [92] Wireshark team. Tshark. <http://www.wireshark.org/docs/man-pages/tshark.html>.
- [93] Wireshark team. Wireshark. <http://www.wireshark.org/>.
- [94] Philip S. Tellis, Steve McAndrew Smith, Michaël Kamp, Wayne Parrott, and Ray Van Dolson. Libyahoo2: A c library for yahoo! messenger. <http://libyahoo2.sourceforge.net/>.
- [95] Edward C. Tolman. Cognitive maps in rats and men, July 1948. <http://psychclassics.yorku.ca/Tolman/Maps/maps.htm>.
- [96] Ryan Trost. *Practical Intrusion Analysis*. Addison-Wesley, 1st edition, 2009.
- [97] Gianluca Varenni, Loris Degioanni, Fulvio Rizzo, and John Bruno. Windump: tcpdump for windows. <http://www.winpcap.org/windump/>.
- [98] Gianluca Varenni, Loris Degioanni, Fulvio Rizzo, and John Bruno. Windump: tcpdump for windows manual. <http://www.winpcap.org/windump/docs/manual.htm>.
- [99] Gianluca Varenni, Loris Degioanni, Fulvio Rizzo, and John Bruno. Winpcap: The windows packet capture library. <http://www.winpcap.org/>.
- [100] S. Waldbusser. Host Resources MIB. RFC 2790 (Standard), March 2000. <http://www.ietf.org/rfc/rfc2790.txt>.
- [101] WhatIsMyIP.com. Whatismyip.com. <http://whatismyip.com>.
- [102] Craig Williams. Exploring a java bot. Article, 2009. [http://blogs.cisco.com/security/comments/exploring\\_a\\_java\\_bot\\_part\\_1/](http://blogs.cisco.com/security/comments/exploring_a_java_bot_part_1/).

- [103] Craig Williams. Exploring a java bot, 2009-2010. [http://blogs.cisco.com/security/comments/exploring\\_a\\_java\\_bot\\_part\\_1/](http://blogs.cisco.com/security/comments/exploring_a_java_bot_part_1/), [http://blogs.cisco.com/security/comments/exploring\\_a\\_java\\_bot\\_part\\_2/](http://blogs.cisco.com/security/comments/exploring_a_java_bot_part_2/), [http://blogs.cisco.com/security/comments/exploring\\_a\\_java\\_bot\\_part\\_3/](http://blogs.cisco.com/security/comments/exploring_a_java_bot_part_3/), [http://blogs.cisco.com/security/comments/exploring\\_a\\_java\\_bot\\_part\\_4/](http://blogs.cisco.com/security/comments/exploring_a_java_bot_part_4/).
- [104] Robin Wood. Kreiosc2 - proof of concept bot. <http://www.digininja.org/kreiosc2/>.



# Appendix A

## Definitions

In order to discuss and better understand botnets, I will introduce some key terms.

A *Botnet* is a collection of computers that are infected with malicious code without the users knowledge or permission and can be remotely controlled through a C2 infrastructure also known as Command and Control (C2). This is also abbreviated to; C&C, CAC or CnC. [62, p. 2]

The term *bot* is an abbreviation of *robot*, and is often referred to as a *zombie* when part of a Distributed Denial of Service (DDoS) attack. A bot is typically a component in a Trojan horse or Back Door system that exploits and controls an individual computer. Most, but not all, bots or Trojans load from the *RUN* key in the windows registry, in order to make sure it is loaded after restarting the machine:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

If the *bot* is self-propagating, it will take on the characteristics of a worm. The bot will participate in a botnet and carry out the commands of the botnet administrator. [62, p. 2] [4, p. 308]

A botnet can be controlled by a single user who own and run it, often referred to as a *botnet admin*, *botnet administrator*, *bot herder*, *bot controller*, *botnet*

*operator* or *botmaster*.

The *IRC protocol* is a text based protocol which is designed to function as a conferencing system. It allows its users to communicate in real time, where the server forms a central rendezvous point where clients (or other servers) can connect to. [66]

*Rootkits* are pieces of software that embed themselves deep within the inner circle of the system. In UNIX systems the system administrator's account is often referred to as the "root" account. Once installed, they feed the systems false information, leak sensitive data and hand over control of the system to an outside attacker. The main services of a *rootkit* are: Concealment, Command and Control (C2) and Surveillance. [8, p. 8-11] Remaining undetected may be the most important service and some rootkits often feed built-in Windows commands such as `netstat.exe` [50] and `TASKLIST.EXE` [53] false information or remove their own process information in order to remain hidden. [9, p. 28-29]

The term *Process-to-port mapping* is the process of finding which process, or application, that is bound to a specific port. For example, applications using the IRC protocol such as the *mIRC* [39] client typically use port 6667. Mapping the communicating ports to their owning application may help to identify applications that are communicating with the Internet, which are not expected to do so.

*Malware*, short for malicious software, is software designed to compromise a computer without the user's knowledge or permission. It is a general term used to describe a variety of different types of unwanted programs, such as viruses, spyware, adware, Trojan horses, botnets, worms and rootkits to name a few.



# Appendix B

## Software

“Any tool is a weapon if you hold it right.”—Ani DiFranco

One can ask the question: What is the difference between *simple* and *elegant*. In my opinion, *simple* is simple. *Elegant* however is simple on the surface and complex underneath. The point being that a simple visualization can be very informative if there is substance to the visualization. It does not have to be complex in order for the data to be shown.

The developed software is mainly divided into two parts. The Process2Port library handles retrieval of the needed information and storing these mappings for the search functions that are to be used by the visualization. The Visualization application uses the created Process2Port library and imported libraries to visualize the data. In this chapter I will explain in more detail the functionality of the different parts of the created library and Visualization as well as existing functionality that is utilized.

### B.1 Starting point

The starting point for the software was the Processing application introduced in Section 3.3.1 and the Carnivore packet capture library for Processing, introduced in Section 3.3.1. Processing provides an easy way to produce nice visuals and

since it is Java based it is also easy to develop Java libraries that can interact with Processing.

The Carnivore library is the main basis for the visualization as it provides packet capture and some readymade examples that give the desired visual effects. In addition it was easy to modify the code to meet changes. At this point it would also be useful to point out what I used from existing sources.

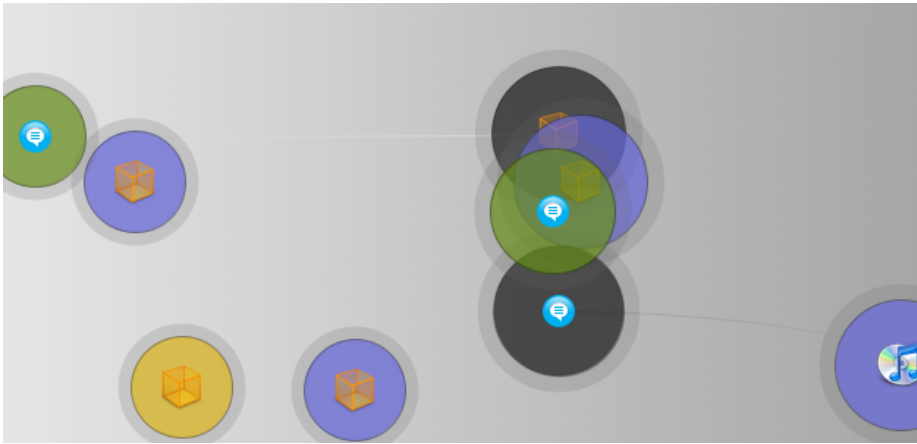


Figure B.1: Original visualization provided by Carnivore library. [23]

In Figure B.1 the basis visualization as provided by examples on the carnivore website are shown. [23] As one can see, the nodes contain image icons for Skype, iTunes and unknown applications. The logic behind the base classification is; selecting a particular image icon based on the specific port number of the packet received or sent. Ports also correspond to a particular color. The placement of the nodes are decided by the last two IP address bytes, and translated into x/y coordinates.

From the example provided by Carnivore, I kept the way nodes are displayed. The node size and ripples are the same, and each node contains an image icon as well as ports corresponding to a particular color.

## B.2 Process2Port

This library is meant to help with a number of different challenges, but the main focus is on doing *process-to-port* mapping, in which communicating ports are mapped to its “owning” application. That means that if an application uses P2P technology, which typically uses a wide range of different ports, the communication can now be tracked back to the application responsible for the connections. I will not go into too much detail with regards to the actual code as it is in excess of 6000 lines, not including 2500 lines of Unit Test code. The package diagram for the Process2Port library can be seen in Figure B.2. The Process2Port library is also packaged with the FatJar eclipse plug-in [25].<sup>1</sup>

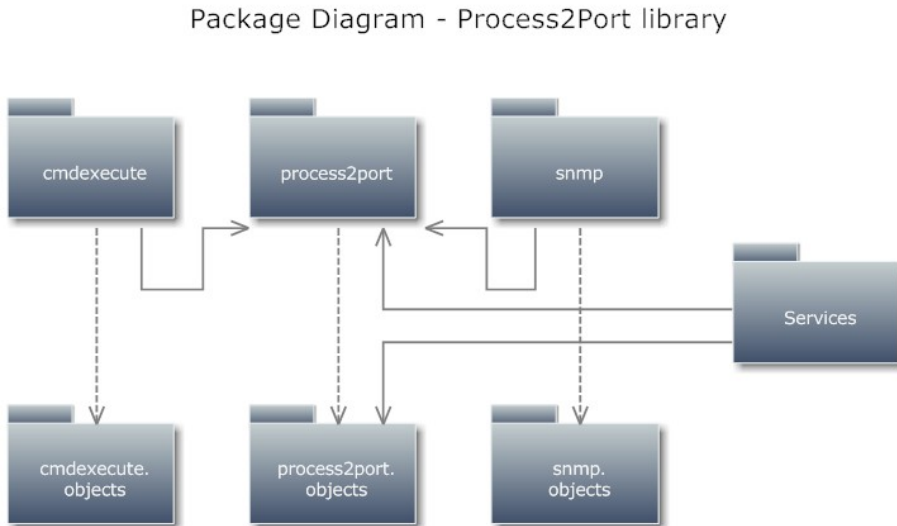


Figure B.2: Process2Port package diagram.

<sup>1</sup>The source code is included with the DVD.

## B.2.1 process2port.objects package

This library contains the main objects used by the process2port package and is meant to contain all the information needed for use with the Process2Port library.

- *IPAddress*: This object is based on the Carnivore library's *IPAddress* object in order to help with interoperability, but it has some added features. The main difference here is an extended functionality from the *java.net.InetAddress.getHostName()* method. In addition to allowing the standard method I have added extended reverse DNS capabilities. This will be further explained in Section B.2.4.
- *Process2Port*: This object contains TCP connection and *HrSWRunEntry* information in one object. It has a date stamp as well as an extended range of "get" functions to help push the contained information to the user. This includes printing the contained information to CSV and Extensible Markup Language (XML).
- *TcpConnectionEntry*: This object must not be confused with *TcpConnectionEntry* SNMP table mentioned in Section B.2.5. It is meant to be a common container of TCP connection information from netstat, TCP-MIB (*TcpConnEntry*) and TCP (Request for Comments (RFC)-793) [71].

## B.2.2 process2port package

This package contains the main functionality and for *process-to-port* mapping. *Process2PortHelper* is used for this purpose.

- *Process2PortHelper*: This is the primary class of the library and handles the *process-to-port* mapping.
- *Process2PortFile*: This class saves the information contained within the *Process2Port* objects into a file using CSV format. In addition it also loads information from a saved file and puts the information into objects. This is for improving performance and accuracy if a *process-to-port* mapping has been carried out previously and the information is still valid.

## Process2PortHelper

The *Process2PortHelper* class can either be started to update the process to port mapping every 30 seconds, as soon as the update is done or at startup. This is because the method updating the Process2Port mapping List may take between 6 and 12 seconds depending on whether *HrSWRunEntryHelper* and/or *SnmpTcpConnEntryHelper* are called. Querying the SNMP agent takes up the majority of the updating functionality.

All update features, HashMaps and Lists can be called at any time by the Visualization application, thus these features are kept synchronized wherever possible. This does unfortunately introduce some latency. The main methods used by the Visualization, is stating how often one wish to update the mappings and the *getActiveproces2portByLocalIPPortPair* and *getActiveproces2portByRemoteIPPortPair* methods. Input for these methods is the connection IP and port as well as a distinction to whether the IP and port pair is the local or remote address.

### B.2.3 cmdexecute package

This package contains the logic for executing commands in a Command Shell and retrieval of specific data such as netstat information, ipconfig and some functionality for killing off processes. The latter is however not implemented in the current version of the Visualization application. Another note is the possibility to execute WMIC commands as an alternative to SNMP, but to focus on the use of SNMP by using *CmdExecConnector*.

The most important classes in this package are:

- *CmdExecConnector*: Executes commands in a Command Shell, returns the output and closes the Command Shell after completion.
- *CmdExecConnectorNewWindow*: Opens a new Command Shell that the user can see and executes commands.
- *DisplayDNSHelper*: Uses *CmdExecConnector* to execute: `ipconfig /displaydns` and returns the values as a *DNSEntry* object. This information is from the internal system mapping of IP to host name. The purpose is to use this with reverse DNS to cut down on execution time and improve accuracy since not all host's respond to reverse DNS queries.

- *NetStatHelper*: Uses *CmdExecConnector* to retrieve netstat information, as mentioned in Section 2.6.5. This is the main component used for mapping IP and port pairs to a specific Process IDentifier (PID). This was implemented after the SNMP, see Section 2.6.5, OID needed for this mapping proved to be difficult to retrieve.
- *ProcessKiller*: Uses *CmdExecConnector* with the command `taskkill /f` to kill processes by their image name or PID.

## B.2.4 services package

This package contains utilities and helper methods used by the other classes in the library. Methods for date stamps, reverse DNS, Timer, TCP and IP header information extraction and other miscellaneous methods. These classes can be accessed by the Visualization application. For example, *TcpHeaderHandler* and *IpHeaderHandler* are used by the Visualization to implement rules based on Sequence Numbers. See Section 2.6.2 for more information regarding TCP and IP headers.

- *CalendarUtils*: This class is used to retrieve the current Time and Date based on *java.util.Calendar*. It is used throughout the library for save files, date stamps for objects and debugging.
- *IPUtilities*: This class performs reverse DNS (*jndi.dns.DnsContextFactory*) in conjunction with *DisplayDNSHelper*, mentioned in Section B.2.3. Additional methods include extracting IP addresses based on REGular EXpressions (REGEX).
- *Timer*: Simple timer used to do performance analysis on particular methods.
- *TcpHeaderHandler*: Based on the JPCAP [13] library with minor modifications to aid in retrieving values. Was implemented because Carnivore uses the same library and this has caused some problems. The class provides an easy way to get access to the TCP header fields.
- *IpHeaderHandler*: Based on the JPCAP [13] library with minor modifications to aid in retrieving values. Was implemented because Carnivore uses the same library and this has caused some problems. The class provides an easy way to get access to the IP header fields.

## B.2.5 snmp package

This package contains the logic for connecting to a SNMP agent and retrieving data. The package also contain helper-methods for retrieving information from *hrSWRunEntry*, *tcpConnTable* as well as *tcpConnectionEntry* as shown in Section 2.6.5. Due to challenges with *tcpConnectionEntry*, netstat was used to map *hrSWRunEntry* and *tcpConnTable*. It is important to note that a SNMP agent needs to be running on the host system in order for this functionality to work. For installation instructions on Win7 please reference Appendix C.

- *SnmpConnector*: Connects to the SNMP agent and retrieve data through *SNMP Get* or *SNMP Walk*. Current SNMP versions supported are *version1* and *version2c*. Useful OID's are also explicitly written here for ease-of-use.
- *HrSWRunEntryHelper*: This class retrieves the information from *hrSWRunEntry* mentioned in Section 2.6.5. It stores the information in a *SnmpHrSWRunEntry* object and puts them in a HashMap where the key is the PID of the entry.
- *SnmpTcpConnEntryHelper*: This class retrieves the information from *TcpConnEntry* mentioned in Section 2.6.5. It stores the values in a *TcpConnectionEntry* object, mentioned in Section B.2.1, and stores the objects in a *List*.
- *SnmpTcpConnectionEntryHelper*: This class retrieves the information from *TcpConnectionEntry* mentioned in Section 2.6.5. Because of problems with retrieving the information from this table the class is not finished, but following the same pattern used by the other classes in this package one can easily implement this table. More information about why there is a problem retrieving the particular SNMP values are available in Section 2.6.5.

## B.3 Visualization

I will focus on the main functionalities provided by the Visualization application. The Visualization handles geolocation of the packets with the use of “Maxmind - geolite City” database [42] shown in Chapter 3.2.1. The Carnivore library is used for some visuals and packet capture. The image used as the background is from the NASA, *Blue marble* project [60] as shown in Figure 3.1 from Section 3.2.1.

In order for the Carnivore packet library to function, winPcap [99] also needs to be installed on the host machine. As with the *process-to-port* mapping library, I will not go into too much detail with regards to the actual code as it is in excess of 2500 lines of code.<sup>2</sup>

A screen shot of the Visualization application in live mode is shown in Figure B.3. The images in the picture also referred to as nodes, correspond to applications running on the local host machine and the location of the nodes are determined based on the location of the IP address it is communicating with.

The links to each node corresponds to the number of packets associated with the connection. There might be more than one link to each node, but as stated in Section 2.6.4 it is problematic to find out if the application receives or sends the packet. In the context of this thesis this is not of paramount importance since it is the connection itself which is the interesting part.

The Visualization shows uTorrent, skype, chrome, msn messenger and spotify. Some of the nodes are not mapped, I.E. the computer images correspond to connections owned by “System idle” and orange boxes are connections for which there is no mapping.

### B.3.1 Key commands and functionality

The Visualization has a number of hot keys to be able to manipulate the image and functionality of the Visualization in real time. Without going into too much detail I will explain the main functionality that the Visualization offers.

#### Navigation

Functionality for zooming and panning the image is provided to the user, with the ability to focus on specific parts of the visualization. This can be done while the Visualization is running. An example of this functionality is shown in Figure B.4. All the hot keys for navigation are shown in Table B.1.

---

<sup>2</sup>The source code is included with the DVD.



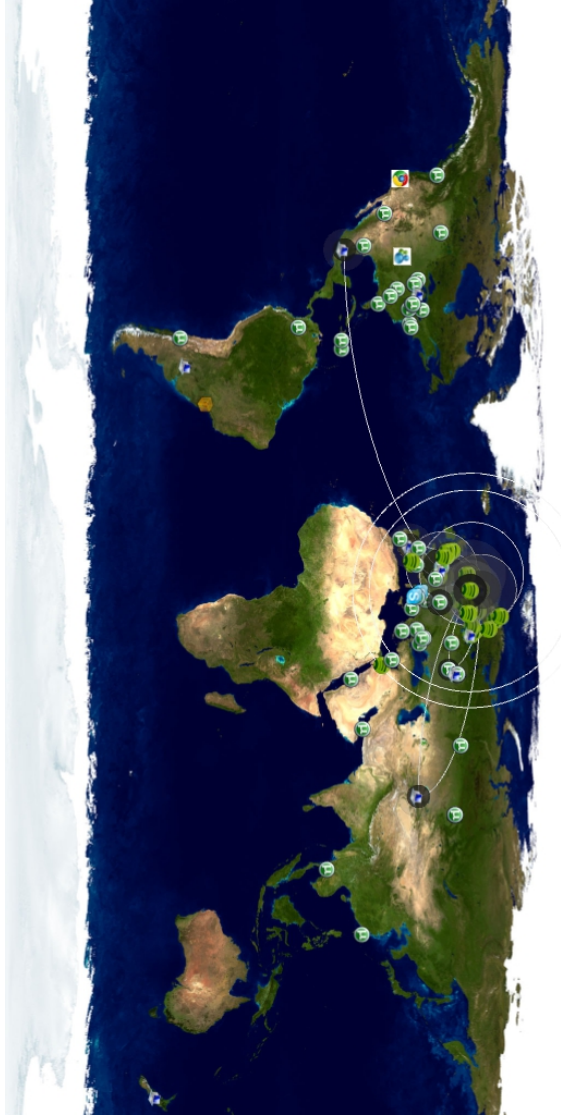


Figure B.3: Visualization after roughly 4 minutes running time. Running Peer-to-Peer (P2P) applications at time of visualization include Skype, Spotify and uTorrent.

<b>Key</b>	<b>Description</b>
<i>Navigation Arrow keys</i>	
UP/DOWN	Move image UP & DOWN
LEFT/RIGHT	Move image LEFT & RIGHT
<i>Navigation Num Pad keys</i>	
+	Zoom inn factor 10
-	Zoom out factor 10
*	Zoom inn factor 100
/	Zoom out factor 100
0	Zoom RESET
8	Move image UP
2	Move image DOWN
4	Move image LEFT
6	Move image RIGHT
7	Move image UP LEFT
9	Move image UP RIGHT
1	Move image DOWN LEFT
3	Move image DOWN RIGHT
5	Move image RESET
<i>Navigation Mouse wheel</i>	
UP/DOWN	Zoom in & out

Table B.1: Visualization key commands



Figure B.4: Zooming and panning the map. Here showing an unknown connection.

### **Setup functionality**

These commands focus primarily on the logging functionality in Processing, as shown in Figure B.5. Information about TCP flags, current time, IP and port pairs and number of unique nodes currently displayed in the Visualization are shown in the Figure B.5 log. Debugging commands are mainly used for showing information from the “headerrules” rule. By enabling the “Image overview table”, all unique mapped applications will be displayed in a list, as shown in Figure B.6. All the hot keys for setup functionality are shown in Table B.2.

### **Various functionality**

These commands manipulate the image while the Visualization application is running. After running the Visualization for a longer period of time, the number of connections shown can clutter the overall picture. The ability to reset the tables containing the nodes helps to keep the picture clean. Functionality to save images, with the possibility to remove all stamps, can be useful for saving

```
[CarnivoreP5] newCarnivorePacket from 87.227.117.86
[Vis-Process] 129.241.208.163:16049 > 87.227.117.86:50121, Date: (19:59:35:695),
Nodes[217], Flags: Flags: [ PSH ACK ]
[CarnivoreP5] newCarnivorePacket from 129.241.208.163
[Vis-Process] 87.227.117.86:50121 > 129.241.208.163:16049, Date: (19:59:35:697),
Nodes[217], Flags: Flags: [ ACK FIN ]
[CarnivoreP5] newCarnivorePacket from 87.227.117.86
[Vis-Process] 129.241.208.163:16049 > 87.227.117.86:50121, Date: (19:59:35:697),
Nodes[217], Flags: Flags: [ ACK FIN ]
[CarnivoreP5] newCarnivorePacket from 129.241.208.163
[Vis-Process] 87.227.117.86:50121 > 129.241.208.163:16049, Date: (19:59:35:698),
Nodes[217], Flags: Flags: [ ACK ]
[CarnivoreP5] newCarnivorePacket from 213.21.83.28
[Vis-Process] 129.241.208.163:53275 > 213.21.83.28:37450, Date: (19:59:35:755),
Nodes[217], Flags: Flags: [ PSH ACK ]
[CarnivoreP5] newCarnivorePacket from 129.241.208.163
[Vis-Process] 213.21.83.28:37450 > 129.241.208.163:53275, Date: (19:59:35:756),
Nodes[217], Flags: Flags: [ PSH ACK ]
[CarnivoreP5] newCarnivorePacket from 129.241.208.163
[Vis-Process] 129.241.208.233:64046 > 129.241.208.163:16049, Date: (19:59:35:759),
Nodes[217], Flags: Flags: [ ACK ]
[CarnivoreP5] newCarnivorePacket from 212.251.141.91
[Vis-Process] 129.241.208.163:53678 > 212.251.141.91:30848, Date: (19:59:35:779),
Nodes[217], Flags: Flags: [ PSH ACK ]
[CarnivoreP5] newCarnivorePacket from 213.100.42.221
[Vis-Process] 129.241.208.194:42042 > 213.100.42.221:50118, Date: (19:59:35:783),
Nodes[217], Flags: Flags: [ SYN ]
[CarnivoreP5] newCarnivorePacket from 87.227.117.86
[Vis-Process] 129.241.208.163:16049 > 87.227.117.86:50121, Date: (19:59:35:799),
Nodes[218], Flags: Flags: [ ACK ]
[CarnivoreP5] newCarnivorePacket from 87.227.117.86
```

Figure B.5: Carnivore console log showing IP and port pairs as well as TCP Flags.

Key	Description
<i>Setup commands</i>	
d	Enable Debugging
l	Enable light Debugging
z	Enable Zooming functionality
g	Enable show debug flags
u	Enable an Image Name Overview table

Table B.2: Setup key commands

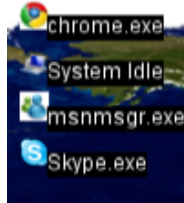


Figure B.6: Visualization showing a list of mapped applications.

interesting findings. All the hot keys for the “various functionality” are shown in Table B.3.

Key	Description
<i>Various commands</i>	
f	Flush memory, reset HashMaps. (Will remove nodes)
s	Save out screen file
t	Toggle Stamps

Table B.3: Various commands

### Rule functionality

All the hot keys for the rules functionality are shown in Table B.4.

Key	Description
<i>Rules commands</i>	
p	Enable remove Unwanted IP (Not my host IP)
i	Enable backup Icons
o	Enable Process2Port
h	Enable TCP IP header fields

Table B.4: Rules commands

### B.3.2 Visualization functionality

The metrics used for the Visualization application is reduced and manipulated with the use of *abstraction layers*. With each added command it is possible to reduce the amount of detail shown to the user. This is vital for creating an overall picture of what is happening, but it can also mean that some useful information could be lost.

#### “Raw” packet captures

The core functionality of the Visualization application is displaying “raw” packet captures, as shown in Figure B.7. All the packets that are caught by the network interface are shown as small orange boxes, or nodes. Each connection that is made is divided into two nodes. The host computers public IP is shown to aid the user with distinguishing what packets are meant for the host and what packets are destined for other computers on the network. As well as putting the information in context when investigating further what information a particular node has. The time of the packet capture as well as the system time is shown to illustrate any latency the Visualization might have. Each node can also be color-coded to emphasize attributes for that particular connection. The color-coding is based on the port used by the individual IP, which was standard from the Carnivore library’s example. [23] The “host IP”, “System time” and “Capture time” tags can be hidden, or toggled, by pressing the “t” key shown in Table B.3.

#### Host IP filtering

The next step in creating a better overview picture is removing packets on the network that is not destined for the host computer, as shown in Figure B.8. Packets are sorted out based on the information in the “host IP” label and this may constitute a significant difference if the network has a high traffic volume. When testing the Visualization application on the NTNU network, a large number of broadcast packets were seen on the network. The overall picture was significantly improved after activating this functionality. It can be activated by pressing the “p” key shown in Table B.4.

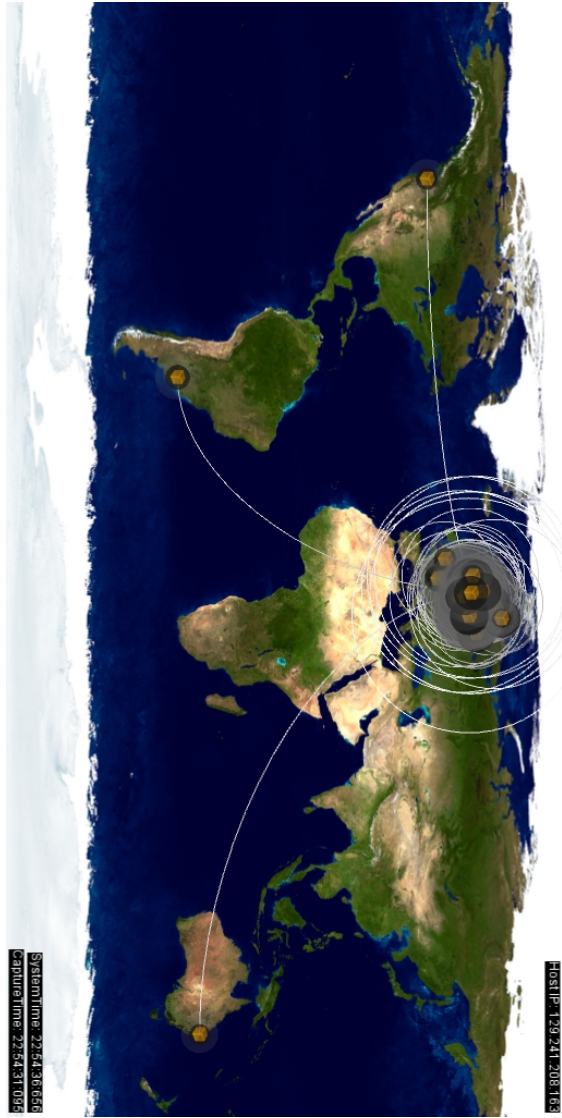


Figure B.7: Visualization of a “raw” packet capture.

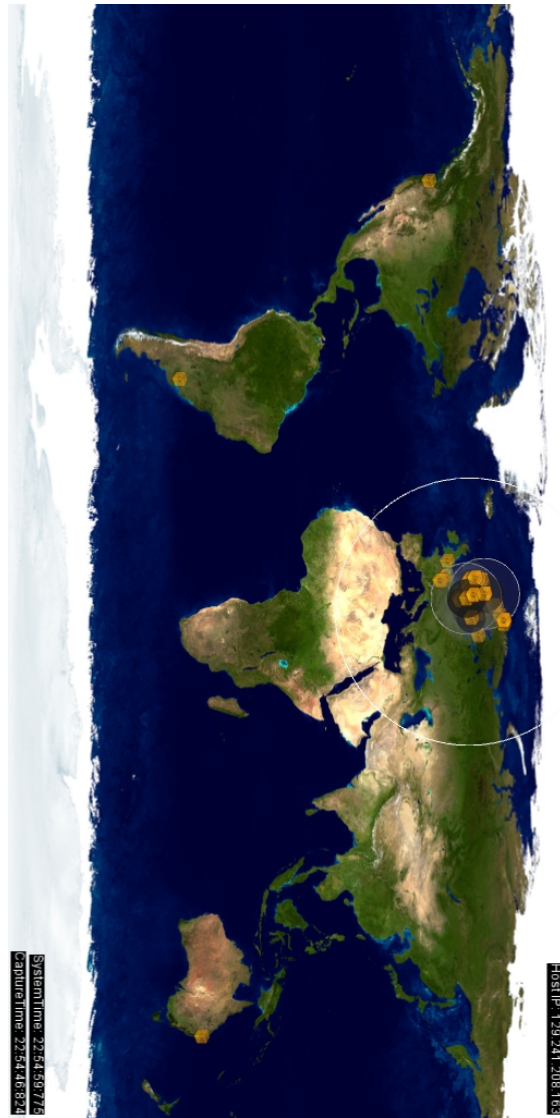


Figure B.8: Visualization with host IP filtering resulting in reduced packets.



## Image icon corresponding to port number

*Image icon to port number* mapping, as shown in Figure B.9, gives each node an appropriate icon image based on the port number used by the IP. This was also part of the original material from Carnivore, but it was extended somewhat to account for applications that was used in the testing. It can be activated by pressing the “*i*” key shown in Table B.4. The potential problem with this kind of mapping is that the port number is mapped to the application that **probably** was responsible for the connection.

If a bot using port 6667 was running on the host computer and the mapping of port 6667 corresponded to *mIRC* [39], the Visualization would falsely show the connection coming from the *mIRC* [39] client. Hence, it would show the connection as part of a legitimate application.

## *Process-to-Port* mapping

As mentioned in Chapter A, *process-to-port* mapping is used to map the connection to its owning application. This was done with the use of SNMP and netstat [50]. When this functionality is activated, by pressing the “*o*” key shown in Table B.4, the *Process2Port* library will create a list of mapped objects with the use of SNMP and netstat [50]. The corresponding image icon will then be mapped to the connection, as seen in Figure B.10.

At the startup of the Visualization, a file of previously mapped connections can be loaded and new connections are saved continuously to the folder: `data\saveCSV\`. The default filename for this CSV file is *process2port.csv* and the format is: Date stamp, Local IP address, Local port, Remote IP address, Remote port, TCP connection state, TCP connection object type, Process Identifier (PID), *Name* of the running application, *Path* of the running application, status of the running application, and type of software the running application is. Not all values may be set in this log file, if the system does not have the information.

Regardless of the node being mapped one can do a reverse DNS, as mentioned in Section B.2.4, on the node in question to determine what host server is behind the IP. In Figure B.10 the connection in question is made to *en.pku.edu.cn*, which corresponds to the “Peking University” in China.

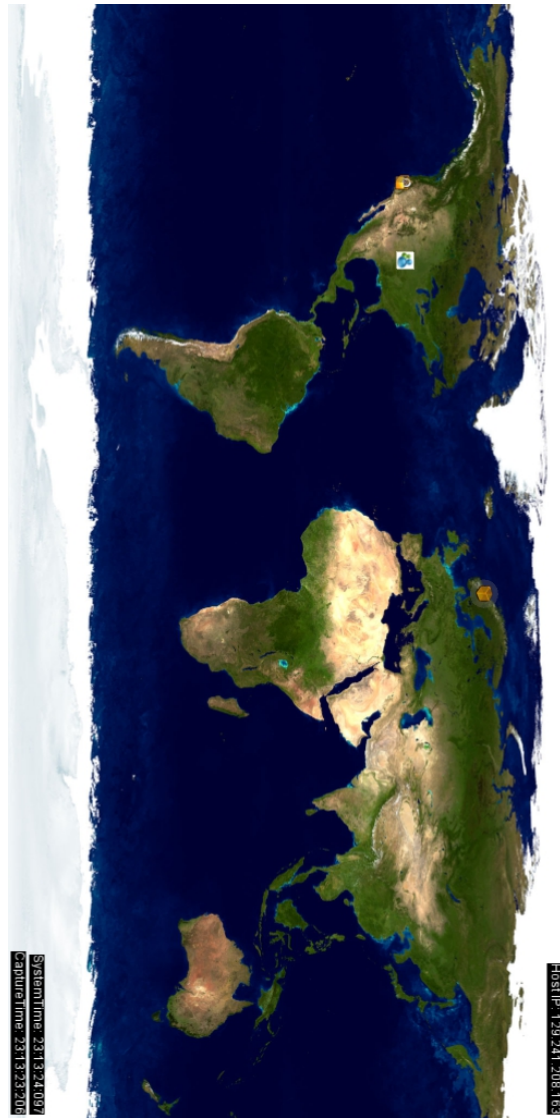


Figure B.9: Visualization showing mapping of image icon to port number.

Location information about the particular node is also shown when selecting a particular node. In Figure B.10 we see that the server is located in Beijing, China. This information can be very useful for putting the existing information into context, as one might see browser activity to a particular location, but not IRC traffic. In this case the “Google Chrome” browser is connecting to the host in China.

### Missing icon

Applications are white listed beforehand to be able to map the correct icon to the corresponding name of the running application. In Figure B.11, *googletalk.exe* is shown to be missing. If dealing with a bot executable not present in the white list, it would appear with “missingIconForImageName”. *Process-to-port* functionality must be present for this to work.

A rule for further reducing traffic, called “Enable TCP IP header fields”, will remove packets that have sequential sequence numbers. This will ensure that the start of the connection will be shown, but the pure data packets that follow will be discarded. These packets will typically have the *PSH* flag set, as mentioned in Section 2.6.2. This functionality can be activated by pressing the “*h*” key shown in Table B.4

To be able to save interesting images one can use the “*s*” key, shown in Table B.3. All the images that shows visualization in this thesis, is from the Visualization application and has been captured with this functionality. These images are saved to folder: `data\saveImage\`

### Image name list

To give a better understanding of what applications are running and communicating on the host computer, the detected running applications are listed with their image name and corresponding image icon. This can be seen in Figure B.12, where the detected running applications can be seen in the upper left corner. A detailed view can also be seen in Figure B.6.

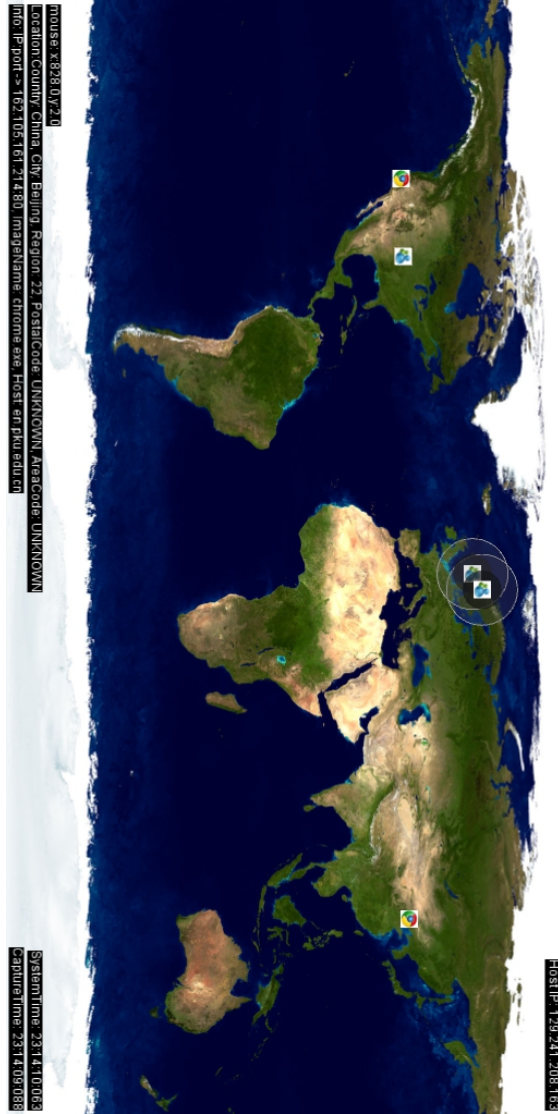


Figure B.10: Visualization showing process to port mapping.

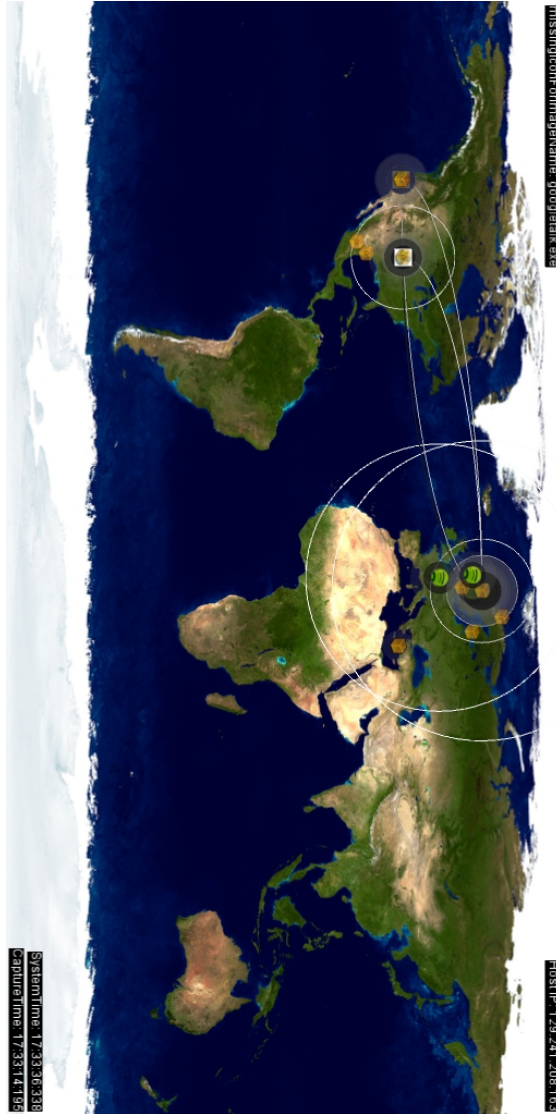


Figure B.11: Visualization showing unmapped applications and resulting in “Missing icon for image name” label.



Figure B.12: Visualization showing list of mapped applications in upper left corner.

### B.3.3 Example usage

When combining these functionalities one can better illustrate the benefits of this kind of visualization. Figure B.13 shows what a typical snapshot looks like. Some interesting observations and questions that have arisen from this figure are:

- All nodes are clustered except one.
- All nodes are in Europe except one.
- Why is my host computer contacting a host in South Africa, Johannesburg?
- Am I currently running *mIRC* [39]? It uses port 6667 and the node is mapped to *mirc.exe*.
- The time is 22:47. Am I at work?

All the nodes in the figure are clustered except one. This is something that is typical for the captures that have been seen when testing the Visualization. Typically the nodes one is communicating with is located in Europe, unless a P2P file sharing application is used. In this figure the majority of the nodes in Europe are mapped to the Spotify steaming music client, which utilize some P2P features. It's quite easy to make out nodes that are not located with the majority of the communication, which can help indicate unusual behavior. In this case one node is located in Johannesburg, South Africa. So one could ask the question; why is the computer contacting a server in South Africa?

The figure also shows that a *mIRC* [39] client has connected to the server in South Africa. Some early botnets such as GT bot, mentioned in Section 2.2, used a hacked copy of the *mIRC* [39] client to communicate with its C2 server. In this case one should ask if the user is running the *mIRC* [39] client. If this application is not running on the computer, then this may indicate that a botnet using the *mIRC* [39] client is running in the background. The server is using port 6667, which is the default port for IRC and used by 56,7% of all botnets as described in Section 2.5.1. This further adds to the validity of the claim that this is an IRC server.

Finally, the time of the capture is also important. According to the “capture time” stamp, the time is 22:47 and assuming that the host computer in question

is a located at the workplace, this could be an employee working overtime or the computer is communicating without a user. The latter could in turn point towards an IRC-based botnet running on the host machine.



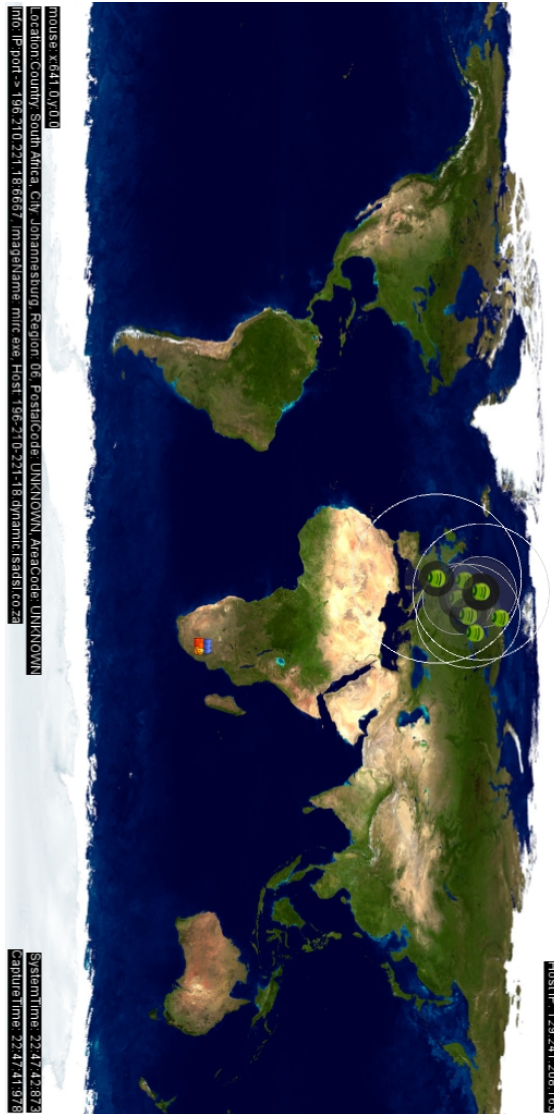


Figure B.13: Example visualization of captured network traffic with mapped mIRC connection.

# Appendix C

## SNMP installation

### C.1 Install SNMP Server on Windows 7

The SNMP service is included with the installation of Windows 7, but you have to set it up yourself.

1. Go to the “Start” button ⇒ “Control Panel” ⇒ “Programs and Features” ⇒ “Turn Windows features on or off”.
2. Navigate to “Simple Network Management Protocol (SNMP)” and “tack off” this and “WMI SNMP Provider”. Follow the installation instructions to the finish.
3. Go to the “Start” button ⇒ “Control Panel” ⇒ “Administrative Tools” ⇒ Open “Services”.
4. Navigate to “SNMP Service” and open.
5. Navigate to the “Agent” tab and fill inn “Contact” and “Location” information. Under the “Service” area make sure that “Physical”, “Applications”, “Datalink and subnetwork”, “Internet” and “End-to-End” is all tacked off. The “Contact” will be the name of the contact person and “Location” is your current location.

6. Navigate to the “Traps” tab and under “Community name” write “public” and click the “Add to list” button.
7. Navigate to the “Security” tab and make sure “Send authentication trap” is tacked off.
8. Under “Accepted community names” click on the “Add...” button.
9. In the “SNMP Service Configuration” window that pops up set “Community rights” to “READ ONLY” and set “Community name” to public.
10. NOTE: If you want to allow write permissions to the SNMP service. Add another “Accepted community name” with “Community rights”, set to “READ WRITE” and set the “Community name” to “private”.
11. In the “Security” tab there is also radio buttons for “Accept SNMP packets from any host” and “Accept SNMP packets from these hosts”. Choose the latter and click the “Add...” button. In the “SNMP Service Configuration” window that opens, and under “Host name, IP or IPX address” field, you write “localhost” and click “Add”. NOTE: Here you can specify which IP addresses you want to be able to connect to the SNMP Service. If you are deploying a distributed monitoring system to include the host machine, add information the address for the central monitoring system here.
12. The service should now be correctly configured and it might be a good idea to restart the system. After the final step it should be possible to access the SNMP service by connecting to the IP 127.0.0.1 on Port 161.

To make sure that the service is up and running as well as enumerating the information in the service, a good program for testing is “Getif”. This will help troubleshoot any problems with the service and help you locate information on the host computer. Note that there is an added MIB package on the website that will allow you to connect and see all the OID’s mentioned in this paper. Another tool package that can be very useful is the net-SNMP package. This is also a great troubleshooting tool and this does not require any add-ons to work.

## **C.2 Install Getif on Windows 7**

Getif [77] is a networking tool written for Windows which collects information from SNMP devices. It makes it easier to discover what kind of information that

resides within the host computer. Download the tool from the Getif [77] website and be sure to download the SNMP4tPC Getif MIB collection also.

# Appendix D

## Excerpt botnet source code

### D.1 a.class

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

class a extends Thread
{
    private File a;
    private byte[] b;
    public static boolean c;

    public a(String paramString)
    {
        try
        {
            a(paramString);
        }
        catch (IOException localIOException)
        {
        }
    }
}
```

```
private void a(String paramString)
    throws IOException
{
    this.a = new File(paramString);
    this.b = new byte[(int)this.a.length()];
    FileInputStream localFileInputStream = new FileInputStream(this.a);
    localFileInputStream.read(this.b);
    localFileInputStream.close();
}

public void run()
{
    while (true)
    {
        if (!this.a.exists())
            try
            {
                FileOutputStream localFileOutputStream = new
                    FileOutputStream(this.a);
                localFileOutputStream.write(this.b, 0, this.b.length);
                localFileOutputStream.flush();
                localFileOutputStream.close();
                if (Main.a != 0)
                    break label62;
            }
            catch (IOException localIOException)
            {
            }
        label62:
        try
        {
            sleep(30000L);
        }
        catch (InterruptedException localInterruptedException)
        {
        }
    }
}
}
```

## D.2 main.class

```
class Main
{
    public static int a;

    public static void main(String[] paramArrayOfString)
    {
        if (paramArrayOfString.length != 1)
            return;
        new a(paramArrayOfString[0]).start();
    }
}
```