Content Agnostic Malware Detection in Networks

Dissertation
zur Erlangung des
mathematisch-naturwissenschaftlichen Doktorgrades
“Doctor rerum naturalium”
der Georg-August-Universität Göttingen

vorgelegt von

Florian Tegeler
aus Osnabrück

Göttingen 2012

Referent:	Professor Dr. Xiaoming Fu
Koreferent:	Professor Dr. Christopher Kruegel

Tag der mündlichen Prüfung: 08. Mai 2012

Abstract

Bots are the root cause of many security problems on the Internet – they send spam, steal information from infected machines, and perform distributed denial of service attacks. Given their security impact, it is not surprising that a large number of techniques have been proposed that aim to detect and mitigate bots, both network-based and host-based approaches.

Detecting bots at the network-level has a number of advantages over host-based solutions, as it allows for the efficient analysis of a large number of hosts without the need for any end point installation. Currently, network-based botnet detection techniques often target the command and control traffic between the bots and their botmaster. Moreover, a significant majority of these techniques are based on the analysis of packet payloads. The proposed approaches range from simple pattern matching against signatures to structural analysis of command and control communication. Unfortunately, deep packet inspection is rendered increasingly ineffective as malware authors start to use obfuscated or encrypted command and control connections.

This thesis presents BotFinder, a novel system that can detect individual, malware-infected hosts in a network, based solely on the statistical patterns of the network traffic they generate, without relying on content analysis. BotFinder uses machine learning techniques to identify the key features of command and control communications, based on observing traffic that bots produce in a controlled environment. Using these features, BotFinder creates models that can be deployed at edge routers to identify infected hosts. The system was trained on several different bot families and evaluated on real-world traffic datasets – most notably, the NetFlow information of a large ISP that contains more than 25 billion flows, which correspond to approximately half a Petabyte of network traffic. The results show that BotFinder achieves high detection rates with very low false positives.

Acknowledgements

I would like to sincerely thank my supervisor Prof. Xiaoming Fu for his constant support, his courtesy to pursue my diverse research interests and the chances he allowed me to take in visiting great research laboratories around the world. His efforts and guidance made this thesis possible.

I am deeply grateful to Prof. Christopher Krügel, who also kindly agreed to be my second thesis supervisor, Prof. Giovanni Vigna and Prof. Richard A. Kemmerer for their help, guidance and the great time I had in the seclab in Santa Barbara.

I would also like to express my gratitude to Prof. Damm, Prof. Hogrefe, Prof. Rieck and Prof. Yahyapour for being a member of my thesis committee.

Last but definitely not least I am deeply grateful to my former and current colleagues at the Computer Networks Group at the University of Göttingen, especially Mayutan Arumaithurai, Niklas Neumann and David Koll. The whole lab helped me to continuously improve through constructive criticism and reviews, hours over hours of discussions, collaboration, and the enjoyable time in the lab.

Contents

Abstract **v**

Acknowledgements **vii**

Table of Contents **ix**

List of Tables **xiii**

List of Figures **xv**

List of Abbreviations **xix**

List of Mathematical Symbols **xxi**

1 Introduction **1**

1.1 Thesis Contribution 4

1.2 Thesis Overview 5

2 Background and Malware Details **7**

2.1 Bots in the Criminal Cyber-Economy 7

2.2 Anti-Virus Scanners 9

2.3 Machine Learning 10

2.4 Bots Under Investigation 11

2.4.1 Banbra 13
2.4.2 Bifrose 13
2.4.3 Blackenergy 14
2.4.4 Dedler 14
2.4.5 Pushdo / Cutwail / Pandex 14
2.4.6 Sasfis 15
2.5 Ant – VirtualBox Malware Environment 15
2.6 Datasets 17
2.6.1 The *LabCapture* Dataset 17
2.6.2 The *ISPNetFlow* Dataset 17

3 Network Traffic Analysis 21
3.1 Available Information 21
3.2 Traces - Chronologically Ordered Connection Sequences 22
3.3 Dimensions of Traces 23
3.4 Basic Features of Traces 23
3.4.1 Average Time Interval 25
3.4.2 Average Duration of Connections 25
3.4.3 Average Number of Source and Destination Bytes Transmitted 26
3.5 Malware Traffic 26
3.5.1 Minimal Trace Length 26
3.5.2 Identification of *C&C* Traffic – Purification 28
3.6 Comparison of Malware Traffic with Benign Network Traffic . . 29

4 BotFinder Design 31
4.1 Input Data Processing 33
4.2 Flow Reassembly 33
4.3 Trace Extraction 34

4.4 Feature Extraction 34
4.5 Model Creation via Clustering 37
4.5.1 The CLUES Clustering Algorithm 37
4.5.2 CLUES Compared to k-means 39
4.5.3 The Quality Rating Function 41
4.5.4 The Final Model 42
4.6 Model Matching during Detection 42
4.6.1 Requirement of Minimal Number of Hits 43
4.6.2 Increasing the Scoring Value 44

5 Deployment Considerations, Implementation, and Performance Benchmarking 47
5.1 Targeted Deployment Scenario 47
5.2 Implementation Details 49
5.2.1 Input Data 49
5.2.2 Flow Reassembly 51
5.2.3 Trace Extraction and Feature Analysis 51
5.2.4 Calculation Client 54
5.2.5 Create Model – Clustering 54
5.2.6 Compare to Model – Detection 56
5.3 Performance 56

6 Parameter Analysis 61
6.1 Minimal Cross-Validation Experiment 61
6.2 Data Purification and the Minimal Trace Length 63
6.2.1 Matching IP Addresses 64
6.2.2 Trace Length and Purification Correlation 66
6.2.3 Detection Rate and False Positives 66
6.2.4 Purification Independence and Cluster Quality 67

6.3 The Cluster Quality Parameter 69
6.4 The Minimal-Hits-Requirement 70
6.5 Trace Input Error Awareness 71
6.6 Overall Impact 72

7 Evaluation 73
7.1 Training BotFinder 74
7.2 Cross Validation 75
7.3 Comparison to *Bothunter* 76
7.4 ISPNetFlow Analysis 78
7.5 Contribution of Features towards Detection 81

8 Bot Evolution 83
8.1 Adding Randomness 83
8.2 Introducing Larger Gaps 85
8.3 High Fluctuation of C&C Servers 87
8.3.1 Recombination 87
8.3.2 Sub-Trace Distribution 88
8.3.3 Computational Complexity 89
8.3.4 Real World Impact 92
8.4 P2P Bots 93
8.5 Bot-like Benign Traffic 94
8.6 Discussion 94

9 Related Work 97

10 Conclusion 101

Bibliography 103

List of Tables

2.1 Evaluation datasets. 17

6.1 Malware families used for training. Purification is especially successful if many traffic samples show traces matching blacklisted IPs. A high quality indicates a low standard deviation within the clusters. 65

6.2 Number of traces available for clustering and the clustering quality. 67

7.1 Detection rate and false positive results of BotFinder (acceptance threshold a=1.8) in the cross-validation experiment and compared to *Bothunter*. 77

7.2 Top-5 aggregated clusters of not blacklisted destination IP addresses. 81

8.1 Worst case amount of required combination steps using n subtraces. 90

List of Figures

2.1 Example of a common malware infection path. After the initial infection, additional malware modules or third party bots are downloaded and installed. 9

2.2 The Ant malware execution environment. 16

2.3 A typical NetFlow data-collection scenario. 18

3.1 Example of the accumulated information in a trace. 23

3.2 Traces with different statistical behaviors (in time and duration space). 24

3.3 Trace length distribution. 27

3.4 Feature relative standard deviations of malware and benign traffic samples: The more periodic a feature, the lower the bar. . . . 30

4.1 General architecture of BotFinder. During the training phase (a), malware samples are run and models are created based on the statistical features of the bots' network behavior. During detection (b), BotFinder analyzes NetFlow or full traffic captures and compares the extracted network features to the established models. 32

4.2 Illustration of the k-means clustering algorithm. 39

4.3 Within sum of squares of k cluster for the average time feature. 40

4.4 Quality rating function by relative standard deviation. 41

4.5 Final model structure. 43

5.1 BotFinder deployment scenario in a large scale network. . . . 49

5.2 BotFinder implementation. 50

5.3 XML based representation of the Pushdo model. 55

5.4 Workload distribution among different BotFinder tasks using one CPU core. 57

5.5 Input data processing using file based and in-memory solutions. For Bro input data, the pre-processing using Bro has to be added to BotFinder's processing. 58

6.1 Detection rate with minimal parameter setting. 63

6.2 Trace length distribution. 65

6.3 Detection rates and false positives based on purification method and minimal trace length. 68

6.4 Influence of the cluster quality control parameter β. 69

6.5 Influence of the minimal feature hit requirement h. 70

6.6 Influence of the δ parameter that rates the quality of traces in the detection phase. 71

6.7 Impact of the BotFinder parameters. All false positive values are plotted on a logarithmic scale. 72

7.1 Detection Rate of BotFinder in cross validation experiments. 74

7.2 Detection Rate of BotFinder in cross validation experiments plotted against logarithmic false positives rate. Optimal values approach are found in the higher left region. 75

7.3 Reported infections over time. 79

7.4 Total reported infections compared to the total number of blacklist confirmed infections. 80

7.5 Normalized contribution of the different features towards a successful detection. 81

8.1 Randomization Impact. 84

8.2 FFT detection quality degradation. 85

8.3 Frequency recognition using the FFT power spectrum. Despite increasing randomization [a) 10%, b) 50%] and additional ≈ 10 large gaps in c) and d), the correct time interval of 50 minutes (Frequency=0.02) is identified. 86

8.4 Fraction of source IPs with more than one sub-trace of length between the “Minimal Trace Length” and 49. 88

8.5 Fraction of IP addresses having exactly a given “Number of Sub-Traces”. 89

8.6 Chronologically ordered potential combinations of five sub-traces A to E. 91

8.7 The recombination rate using different thresholds t during pre-processing. 93

9.1 Vertical and horizontal correlation for malware detection. 98

List of Abbreviations

AV	Anti-Virus
CDF	Cumulative Distribution Function
CQ	Clustering Quality
DoS	Denial of Service
DDoS	Distributed Denial of Service
DHT	Distributed Hash Table
DFT	Discrete Fourier Transform
DPI	Deep Packet Inspection
FFT	Fast Fourier Transform
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information Export
IRC	Internet Relay Chat
ISP	Internet Service Provider
MAC	Media Access Control
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NIDS	Network Intrusion Detection System
OS	Operating System

PCA	Principal Component Analysis
PPI	Pay-Per-Install
P2P	Peer-To-Peer
SD	Spectral Density
TOR	The Onion Router
VM	Virtual Machine

List of Mathematical Symbols

α:	Model acceptance threshold during detection and thereby the sensitivity of the system.
β:	Exponent of the cluster quality function
γ:	Scoring value during model matching
δ:	Exponent of the trace quality rating function
σ:	Standard deviation of a dataset
t:	Acceptance threshold for sub-trace combination during the pre-processing step
$\mathcal{M}$:	Set of all models
$\mathcal{T}$:	A trace: A chronologically ordered sequence of connections
$\lvert\mathcal{T}\rvert_{min}$:	The minimal length of a trace

Chapter 1

Introduction

Many security problems on today's Internet such as spam, Distributed Denial of Service (DDoS) attacks, data theft and click fraud are caused by malicious software that runs undetected on end-user machines. A very efficient, and arguably, the most relevant kind of such malware are *bots*. Hereby, the malware opens a *command and control (C&C)* channel [18] to a single entity – called the *botmaster* – and uploads stolen information and awaits new commands. A group of infected hosts reporting to the same botmaster is referred to as *botnet* [16, 25, 61]. These botnets are malicious infrastructures that can carry out a number of different criminal activities that significantly impact the overall security in the Internet.

As the malware's focus and design shifted in the recent years from a fun-motivated hacking idea to a malicious criminal economy [33], bot authors take great care to make their bots resilient against easy detection or removal [23]. Consequently, defenses against malware infections are a high priority in the industry and the security research community whereby identification of infected machines is the first step on the way to purge the Internet of bots.

The traditional way of malware detection is the installation of host-based systems such as Anti-Virus (AV) scanners. Unfortunately, these scanners have the significant drawback that end-users with widespread skill levels in computer administration are in charge of ensuring an up-to-date protection of their machines. Furthermore, an AV scanner is not necessarily able to detect a local infection due to its increasingly stealthy behavior: For example, a Zeus study [75] revealed that of 10,000 real world infections of this financial trojan, 55% occurred on systems with an up-to-date virus scanner installed. In addition, “only” 71% of the overall hosts under analysis had a recent anti-virus solution, 6% an outdated one and 23% had none at all.

In the light of AV installation rates of 71% and the existence of numerous botnets in the wild, complementary solutions to detect malware infections are required. As a consequence, network-based bot detection approaches are increasingly deployed for complementary protection. Such network devices provide a number of advantages, such as the possibility to inspect a large number of hosts without the need for any end-point installation or the ability for the network provider to quickly warn the provider’s clients if their machines are infected.

Existing techniques for identifying bot-infected hosts by observing network events can be broadly divided into two approaches. One approach relies on *vertical correlation*, where network events and traffic are inspected, looking for typical evidence of bot infections (such as scanning) or command and control communication produced by individual hosts [80, 31, 28]. Such approaches typically rely on the presence of a specific bot-infection life-cycle or noisy bot behavior, such as scans, spam, or Denial of Service (DoS) traffic (Silveira et al. [65]). Moreover, they usually search packet payloads for specific signatures, making them unsuited for encrypted *C&C* traffic.

The second approach focuses on *horizontal correlation* of activities carried out by multiple hosts. More precisely, network traffic is examined to identify cases in which two or more hosts are involved in similar, malicious communication [32, 30, 82]. The main strength of these techniques is that they are independent on the underlying botnet structure and do not require access to packet content. However, systems based on horizontal correlations require that at least two hosts in the monitored network are infected by the same botnet, and, in addition, exhibit unique communication patterns that can be correlated. As a consequence, such techniques inherently cannot detect individual bot-infected hosts. This is a significant limitation, especially when considering the trend toward smaller botnets [16]. In addition, horizontal correlation techniques are usually triggered by noisy activity [30]. This is problematic, as the past few years have seen a shift of malware from a for-fun activity to a for-profit one [23]. As a result, bots are becoming increasingly stealthy, and new detection techniques that do not rely on the presence of noisy activities need to be explored.

Starting from the assumption that malware authors will further raise the bar to successful detection by continuing the trend towards encrypted *C&C* communication, this thesis investigates novel solutions for network-level bot detection based on packet header information only. Being payload agnostic is advantageous in many ways: From an Internet Service Provider (ISP)'s perspective it is significantly easier to obtain packet header information than full packet captures. Such streams contain all relevant information about every connection handled on the router. Moreover, end-users privacy is less impacted compared to full traffic captures.

As a consequence of the requirement to ignore packet payloads, the fundamental questions investigated and answered in this thesis are:

1. Does network traffic of bots exhibit any special features that distinguish it from other, benign traffic that a typical end-host emits?
2. If bot traffic can be distinguished from other traffic, can one use this to construct a system that exploits this difference in traffic to detect bots – preferably in an automated way?
3. Is such a system's performance in terms of processing speed and detection quality high enough to provide a valuable complement to existing solutions?

To anticipate the results, all questions can be answered with a clear "yes": In the following, this thesis presents the analysis of network traffic of different malware families, that allow the observation that *C&C* connections associated with a particular bot family follow certain regular patterns. That is, bots of a certain family send similar traffic to their *C&C* server to request commands, and they upload information about infected hosts in a specific format. Also, repeated connections to the command and control infrastructure often follow certain timing patterns.

The regularity in *C&C* network traffic is leveraged to create BotFinder, a vertical correlation system that detects individual, bot-infected machines by monitoring their network traffic. BotFinder works by automatically building models for *C&C* traffic of different malware families. To this end, bot instances that belong to a single family are executed in a controlled environment and their traffic is recorded. In the next step, the system extracts features related to this traffic and uses them to build a detection model. The detection model can then be applied to unknown network traffic. When traffic is found that matches the model, the host responsible for this traffic if flagged as infected.

BotFinder offers a combination of salient properties that sets it apart from previous work. First, it does not correlate activity among multiple hosts during the detection phase. This allows it to detect individual bot infections (or bots that deliberately behave in a non-synchronized way to avoid detection). Second, the system does not require access to packet payloads. Thus, it is resilient to the presence of encrypted bot communication, and it can process network-level information such as NetFlow. This is a significant advantage over related work [80, 31, 28] that explicitly investigate packet contents, which is a computationally intense, sometimes legally problematic, and recently increasingly evaded way of bot detection. Moreover, BotFinder does not rely on the presence of noisy activities, such as scanning or denial-of-service attacks to identify bot-infected hosts.

The BotFinder approach is evaluated by generating detection models for a number of botnet families. These families are currently active in the wild, and make use of a mix of different infection and *C&C* strategies. The results show that BotFinder is able to detect malicious traffic from these bots with high accuracy. The detection models are also applied to traffic collected both on an academic computer laboratory network and a large ISP network with tens of billions of flows. These experiments demonstrate that BotFinder produces promising detection results with few false positives.

1.1 Thesis Contribution

In summary, this thesis makes the following contributions:

- The observation that *C&C* traffic of different bot families exhibit regularities, both in terms of traffic properties and timing, that can be leveraged for network-based detection of bot-infected hosts.
- It presents BotFinder, a learning-based approach that automatically generates bot detection models and does not require packet payload information. Bot binaries are run in a controlled environment and their traffic is recorded. Using this data, models of characteristic network traffic features are build.
- The implementation of BotFinder, that shows that the system is able to operate on high-performance networks with hundreds of thousands of active hosts and Gigabit throughput in real time. The application of

BotFinder to real traffic traces demonstrates its high detection rate and low false positive rate. Additionally, the thesis shows that BotFinder outperforms existing bot detection systems and it discusses how BotFinder handles certain evasion strategies by adaptive attackers.

1.2 Thesis Overview

The remainder of this thesis is organized as follows: In Chapter 2, background on the cyber-crime economy of malware is given, which motivates the development of certain types of stealthier bots. A technical malware analysis framework is presented and the bot families and datasets used in this thesis are introduced. In Chapter 3, traffic obtained from malware samples is compared to "normal" end-host traffic and the concept of *traces* is presented. Based on the observation that *C&C* traces exhibit periodic behavior, the content agnostic malware detection framework BotFinder is created in Chapter 4. Chapter 5 shows the implementation, potential deployment scenarios and a computational performance evaluation of the BotFinder prototype and Chapter 6 investigates the influence of BotFinder's parameters. In Chapter 7, BotFinder's performance with regards to the bot detection rate and false positives is evaluated in a cross-validation experiment, BotFinder is compared to the related work *Bothunter* and BotFinder is applied to the large *ISP-NetFlow* dataset. In Chapter 8, potential detection evasion strategies of next generation malware are discussed and their impact on BotFinder is analyzed. Chapter 9 places BotFinder in context to the related works and Chapter 10 concludes the thesis.

Chapter 2

Background and Malware Details

This chapter gives an overview on general bot functionalities and discusses the economic motivation of malware authors to design their bots in a stealthy and efficient way. Further, it presents the analysis framework Anubis [5] that allows identification of malicious software based on behavioral models, and it introduces the malware families investigated in this thesis. Finally, this chapter presents Ant, a VirtualBox[1]-based malware execution environment.

2.1 Bots in the Criminal Cyber-Economy

Although initial malware development might have been motivated by a kind of "hacker spirit" and the fun to overcome technical barriers, modern malware is typically a fundamental block of cyber crime [33] with the sole purpose to gain substantial profits.

These profits are earned for example by spam, which includes all formes of unauthorized and unsolicited advertising such as email spam, blog post spam [55], Twitter spam [29] or forum spam [64]. The revenue and profit gain of spammers is hard to estimate and typically based on interpolation of observed transactions or internal botnet knowledge [71]. Archak et al. [2] investigated the product markets advertised in spam mails and identified affiliate programs as the core money income source for the spammer. Such programs

[1]`https://www.virtualbox.org/`

pay on commission basis, for example between 30-50% in the pharmaceutical market [63, 44]. Different estimates for the spammer's revenues range from 2-3 millions a year [49, 37] up to high values as 2 million dollars per day for a single botnet [1]. A main – if not the main – source of spam are botnets [12, 45, 62]; for example, in a single study, Xie et al. [81] identified 7,721 botnet-based spam campaigns covering $\approx 340,000$ unique botnet host Internet Protocol (IP) addresses.

Fighting botnets is worthwhile for the spam alone, especially considering that the computer users themselves do not see a personal responsibility to better defend against these threats [35], but hold ISPs and anti-virus software suppliers responsible. However, multiple other income vectors to botnet operators exist, for example click fraud [41] in advertisement networks or the data theft of personal information such as social security numbers or credit card data. To illustrate the significance of the latter, please note that Stone-Gross [71] obtained 1,660 unique credit card numbers during a relatively short period of 10 days of operational control over the Torpig botnet with 180,000 infected members. Together with the blackmailing threat to execute DDoS attacks, botnets offer a variety of tools for malicious cyber-criminals.

As a consequence, the cyber crime economy specializes further and criminals establish underground markets to offer and request services. For example, Caballero et al. [8] illustrate that some malware authors specialized on infecting victim hosts and sell the service to install third-party malware for prices of around \$100-\$180 per thousand unique installs. On the infected hosts, methods as depicted in Figure 2.1 are applied. Here, the initial malware is a very small and carries highly packed infection-code that "opens the door" to the victim host by exploiting security flaws. After initial infection, the bot contacts the *C&C* server and starts to download further malware. This ability is often referred to as *dropper* functionality. Such additional malicious software might either be a module of the malware itself or a completely independent malware downloaded on behalf and for payment (Pay-Per-Install (PPI)) of other participants in the underground economy [71, 70]. Therefore, the initial malware and the follow-up installations may vary on each end host.

The PPI infrastructure offered allows botnet operators that roll out a new botnet version to buy large quantities of infected hosts and then directly start to gain profits by selling their new botnet to spammers or other cyber criminals. Overall, the cybercrime economy is highly sophisticated and the malware in operation becomes more versatile, modular and flexible to exploit all possible revenue vectors.

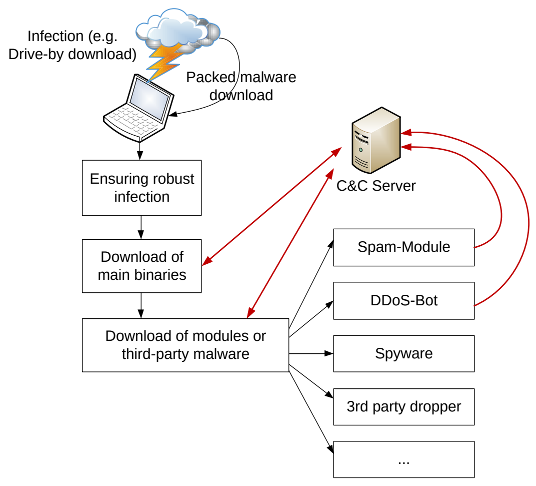

Infection (e.g. Drive-by download)
Packed malware download
Ensuring robust infection
Download of main binaries
Download of modules or third-party malware
C&C Server
Spam-Module
DDoS-Bot
Spyware
3rd party dropper
...

Figure 2.1: Example of a common malware infection path. After the initial infection, additional malware modules or third party bots are downloaded and installed.

2.2 Anti-Virus Scanners

The classical countermeasure against malware are AV-scanners – host base software components that try to detect and remove malware infections of different kinds such as worms, bots and viruses. Hereby, the classification of malware into groups is not always clear and the classes are often "smeared out", as many malware systems offer overlapping functionalities. Typically, modern scanners from Kaspersky[2], TrendMicro[3], AVG[4], Symantec[5] or McAfee[6] also try to prevent infections by scanning the life file access on the end host, suspicious registry access or malicious software behavior (like typical steps to install a root-kit). Recent updates allow the scanners to know binary signatures of most of the currently active viruses. To detect unknown malicious software, heuristics are deployed that rate the behavior or code fragments of end host software. However, AV systems have a number of drawbacks. First, the AV systems run at very high trust levels which opens potential infection paths

[2]http://www.kaspersky.com
[3]http://www.trendmicro.com
[4]http://www.avg.com
[5]http://www.symantec.com
[6]http://www.mcafee.com

and inflicts the system performance by the file and activity surveillance. Second, the AV scanners have to be installed at every end-host that should be protected. In the private sector at end-users, some users are technically unable to ensure a proper and continuously updated anti-virus installation. Third, as mentioned in the introduction, even an up-to-date is not necessarily protecting the system from advanced malware infections like the Zeus bot.

A helpful asset for security researchers performing AV-scanner analysis is the webservice *VirusTotal*[7]: Users submit suspicious executables to the website and receive the accumulated output of, at time of February 2012, 41 anti-virus solutions. If a majority of AV systems report the same malware infection, this can be considered as a good malware classification hypothesis for further research.

2.3 Machine Learning

The general concept of machine learning is the process of knowledge generation or derivation from data by an artificial entity – typically a computer program. Instead of plain reading and storing of data, underlying principles are investigated and general laws that are contained in the data under analysis are derived. In a generic way, Mitchell [51][8] summarized this process in the following definition:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Often, the machine learning process is divided into a learning phase to gain experience and an application phase to perform the task, e.g., a classification of data sets: In the first phase, examples are analyzed and the underlying principles are derived. In the second phase, the system performs the task and applies the learned knowledge. In a data classification example, the learning step might lead to rules to group data sets. These rules are then applied in the second phase to classify unknown data.

Algorithms used in machine learning can be divided into the two domains of *supervised* and *unsupervised* learning. In supervised learning, a human expert provides labels to the data and "guides" the algorithm by input-output

[7]`https://www.virustotal.com/`
[8]Chapter 1, Page 2

specifications. In unsupervised learning, unknown data is investigated under the assumption of minimal to none prior knowledge. Typical techniques in unsupervised learning are clustering mechanisms and Principal Component Analysis (PCA).

In a special type of machine learning – the *reinforcement learning* [73] – the results of the performed tasks generate some feedback that complements the training set in follow-up rounds of learning. Hereby, the algorithm iteratively approaches high quality models for the given challenge. Unfortunately, this approach only works in scenarios where either a human expert is able to provide the feedback or some kind of environment generates feedback that is usable in an automated fashion.

The application space of machine learning is extremely broad and ranges from computer vision over natural language processing to DNA sequence analysis, artificial intelligence and many more. In this thesis, as will be shown in Chapter 4, unsupervised machine learning techniques such as clustering are used for automated security analysis. This application of machine learning is a well suited and often applied approach to tackle the problem of the vast number of malware samples active in the wild. The high number of samples makes it impossible for a human security researcher to investigate every suspicious binary. The amount of samples is even further increased when polymorphic code modification or frequent updates are applied by the malware. The time and cost extensive human analysis can hereby be replaced – or at least be assisted – by automated malware analysis and detection [43, 50].

2.4 Bots Under Investigation

A common problem in detection and classification of malware is the highly versatile binary representation that prevents simple signature matching. Moreover, various incrementally modified or improved versions of the bot exist in parallel. Typically, a malware author modifies and updates its malware by adding new functionality, counteracting recent security advances (e.g., in virus scanners) or changes the communication mechanism. To group such malicious binaries that use shared binary code and typically originate from the same author, the concept of *malware families* is used. Samples that belong to the same family, often share communication fragments in the *C&C* exchange or behave similarly on a technical level.

Throughout this thesis, the classification of malicious samples is based on the combination of the following techniques to identify a given binary sample:

- Using a large number of **anti-virus scanners** in parallel increases the probability of detecting and identifying a given malware sample. As aforementioned, the VirusTotal project yields a reasonable starting guess for further analysis.
- **Behavioral similarity analysis** as, for example, applied in the sandbox environment Anubis [5]: Anubis is a tool that analyzes Windows malware samples (any kind of executable file) submitted from users through a public webpage. As a dynamic malware analysis platform, it is similar to environments such as BitBlaze [67], CWSandbox [78], and Ether[19]). Anubis, which superseeds Bayer et al.'s TTAnalyze [4], executes the received samples in a restricted QEMU [6]-based environment and observes the behavior of the malware. In this realm, behavior especially focuses on security relevant aspects such as modifications made to the registry or the file system, interactions with other processes and the network traffic. The obtained information is typically investigated by a security researcher that judges and classifies the malware sample. Yet, as security companies receive thousands of malware samples a day Bayer et al. proposed a method for automated, behavior based scaling [3]. Hereby, Anubis' reports are used to formalize the malware's behavior and create behavioral profiles that allow clustering. This approach is highly advantageous if a certain amount of malware samples are already classified, as similar behavior indicates that members of the individual clusters belong to the same malware family. Hereby, simple detection evasion techniques employed by malware authors, such as small code variations or modified signatures, are rendered ineffective. If this analysis is complemented with the result of various AV-scanners, Anubis allows automated classification of malware samples.
- **Blacklist comparison** of destination IP addresses contacted by the executed malware sample: If the malware under investigation contacts an already known and malware-attributed malicious server, it is reasonable to assume that other samples contacting the same server also belong to the same malware family.
- **Manual traffic inspection** may allow definite identification of the malware sample by signature analysis and http traffic inspection. However, packet inspection is applicable on unencrypted traffic only.

The following six different malware families are a representative mix of families that are currently active and observed in the wild. More precisely, these families represent active malware samples observed in the UCSB Anubis malware analysis sandbox. Therefore, it is ensured that all following analysis operates on malware that is active and relevant.

2.4.1 Banbra

Banbra is a trojan horse/spyware program that downloads and installs further malware components from the *C&C* server secretly. It specifically targets Brazilian banking websites and employs an interesting strategy to remove protection mechanisms deployed by these banks [9]. For this purpose, it downloads the legitimate malware removal tool Avenger by Swandog[10]. According to the Avenger's website, *"The Avenger is a fully-scriptable, kernel-level Windows driver designed to remove highly persistent files, registry keys/values, and other drivers protected by entrenched malware."*. This toolkit is then (mis)used to remove the protection system from the banking websites and pave the way to allow Banbra to steal banking credentials. To do so, Banbra injects malicious HTML code and logs the user's keystrokes to steal and upload credentials.

2.4.2 Bifrose

The Bifrose family is also represented in the trojan variants called Bifrost and sum up to a family of more than 10 variants of backdoor trojans. These trojans establish a connection to a remote host on port 81 and allow a malicious user to access the infected machine[11][12]. Among other functions, it allows the attacker to log keystrokes, manipulate the registry and execute arbitrary commands. Interestingly, Bifrose is using the The Onion Router (TOR) network [20] (multiple routers that forward onion-like, encapsulated encrypted information and provide anonymity between the sender and the receiver) in an attempt to evade detection by network signatures and, implicitly, enumeration approaches from security researchers. It is thereby a good representative of

[9]`http://www.f-secure.com/v-descs/trojan-spy_w32_banbra_rm.shtml`, 2011-11-28

[10]`http://swandog46.geekstogo.com/avenger2/avenger2.html`, 2011-11-28

[11]`http://www.f-secure.com/v-descs/backdoor_w32_bifrose_bge.shtml`, 2011-11-28

[12]`http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Backdoor%3AWin32%2FBifrose`, retrieved 2011-11-27

encrypted and stealthy communication that effectively evades most classical network based detection mechanisms.

2.4.3 Blackenergy

Blackenergy is a DDoS bot [53] that communicates through HTTP requests. The current version 2 is an advanced evolution from prior versions that explicitly improved in the field of rootkit/process-injection techniques and adds strong encryption and a modular architecture [68]. The infection cycle follows a dropper approach where a small initial infection installs a rootkit and installs itself as a service in Windows. Then it downloads and installs further modules. An interesting aspect with respect to the work presented in this thesis is the encryption used by Blackenergy: Whereas for content, a hard-coded 128-bit key is used, the network traffic is encrypted with a unique identification string as key. This effectively counteracts the risk that all botnet traffic can be decrypted after security researchers have obtained a single key from a controlled installation.

2.4.4 Dedler

Dedler is a classical spambot that is active in different versions (Dedler.AA to Dedler.W, also depending on the AV company) from a simple worm that spreads through open fileshares to an advanced trojan/spambot system. Whereas initial versions appeared already in 2004[13], recent versions are still prevalent and active as seen in the Anubis malware analysis environment.

2.4.5 Pushdo / Cutwail / Pandex

The Pushdo botnet[14] also known as Pandex or Cutwail, is a powerful botnot active in the wild since January 2007[15]. It is a very advanced DDoS and spamming botnet responsible for approximately 4 percent of the overall spam

[13]`http://www.symantec.com/security_response/writeup.jsp?docid=2004-050714-2558-99`, retrieved 2011-11-29

[14]In-depth study by TrendMicro: `http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/study_of_pushdo.pdf`, retrieved 2011-12-08

[15]`http://about-threats.trendmicro.com/ArchiveMalware.aspx?language=us&name=TROJ_PANDEX.A`, retrieved 2011-12-08

volume[16], making it one of the largest spam botnets in the world. With regards to the botnet's name, some confusion originates from the naming of the main binaries. The advanced downloader is called *Pushdo* and the spamming module of Pushdo is named *Cutwail*.

The malware follows a typical infection cycle as depicted in Figure 2.1 by installing multiple levels of protection on the infected host to complicate detection and removal. After the initial Pushdo engine is installed, additional malware such as the Cutwail spam engine is downloaded and executed as modules. Cutwail configures itself and reports the client configuration to the *C&C* server. In the next steps, the server is contacted again and spam content is retrieved. As TrendMicro reports, the spam engine continues to send spam until the entire run has been completed and finally requests a new spam run and sleeps for a period of time. This period of inactive time between *C&C* requests is finally exploited by BotFinder to detect the malware on the network level.

Additional to the spamming, Pushdo loads so-called campaign modules that might contain pop-up advertisements to lure an user into buying fake services or subscribe to malicious websites that again launch multiple attacks on the victim's PC. Another module loaded by Pushdo contains DDoS functionalities.

2.4.6 Sasfis

Sasfis is a trojan horse that spreads via spam or infected web-pages and allows the remote control of compromised machines[17]. Following typical bot behavior, the *C&C* channel is used to transfer new commands or download additional malware to the computer. A final use of Sasfis is to work as a bot-for-hire that allows attackers to install additional malware for a payment.

2.5 Ant – VirtualBox Malware Environment

To execute large amounts of malware samples in parallel, large amounts of VirtualBox[18] Virtual Machine (VM)s are run in parallel which each execute a

[16]`http://www.symanteccloud.com/de/de/download.get?filename=MLIReport_2009.01_Jan_Final.pdf`, retrieved 2011-12-08

[17]`http://www.symantec.com/security_response/writeup.jsp?docid=2010-020210-5440-99`

[18]`http://virtualbox.org`

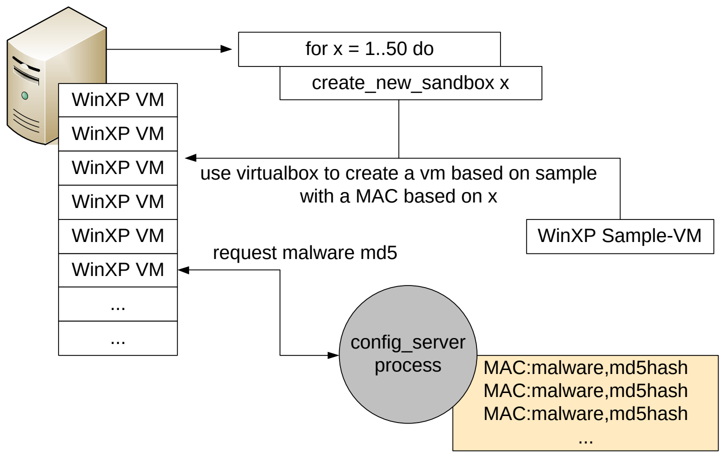


Figure 2.2: The Ant malware execution environment.

single malware sample instance. The process is automated using a set of scripts called *Ant* that allows to start strapped down Windows XP virtual machines and automatically load malware samples from Anubis.

The technical process is depicted in Figure 2.2. To start numerous VMs (50 in the Figure), a basic VM of an operating system sample needs to be prepared. All instances are derived from this initial VM and the only difference between the machines is the Media Access Control (MAC) address. Ant sequentially starts the VMs and automatically iterates the VM-number, which leads to the MAC address. In Figure 2.2 this incremental number is referred to as x. After loading the Operating System (OS), a script starts to run on the VM and requests the malware name and the MD5 hash of the binary to download. The download is performed automatically from the Anubis malware database based on the MD5 hash. To learn about this MD5, the script contacts a configuration server, which runs on localhost (a simple perl script) and reads the MAC-to-malware-matchings from a file and serves it to the VMs. Directly after download, the malware sample is executed in the VM. All network traffic is captured separately for each MAC so that observed traffic can easily be attributed to the corresponding malware sample.

Running Windows XP VMs with minimum requirements allows the parallel execution of up to 50 virtual machines on an eight-core Intel Core i7 CPU (3.07GHz) server equipped with 12 Gigabytes of RAM.

	LabCapture	*ISPNetFlow*
Traffic	$\approx$ 3.6 TB	$\approx$ 540 TB
Internal hosts	$\approx$ 80	$\approx$1M
Concurrently active	$\approx$60	$\approx$250k
Start time	2011-05-04	2011-05-28
Length	84 days	37 days
Connection	$\approx 64.3 \cdot 10^6$	$\approx 2.5 \cdot 10^{10}$
Long Traces	$\approx$ 39k	$\approx$ 30M

Table 2.1: Evaluation datasets.

2.6 Datasets

In this thesis, two main datasets, as shown in Table 2.1, are used to analyze malicious network traffic and compare it to "normal" network traffic from different sources: The *LabCapture* and the *ISPNetFlow* datasets.

2.6.1 The *LabCapture* Dataset

The *LabCapture* dataset is a full packet capture of 2.5 months of traffic of the UCSB security lab with approximately 80 lab machines (including bridged VMs). According to the lab policy, no malware-related experiments should have been executed in this environment, and the *LabCapture* should consist of only benign traffic. Still, some researchers might have ignored the policy and malware traces might exist in the traffic. However, the advantage of this dataset is that it contains the full packets of all communications. This allows to inspect suspicious traffic or potential infections manually.

2.6.2 The *ISPNetFlow* Dataset

The *ISPNetFlow* dataset is a large dataset covering 37 days of unsampled NetFlow v5 data collected from a large network provider serving over one million customers. The captured data reflects around 540 Terabytes of data or, in other words, 170 Megabytes per second of traffic.

NetFlow [13] is a widely used standard by Cisco covering network flow information such as:

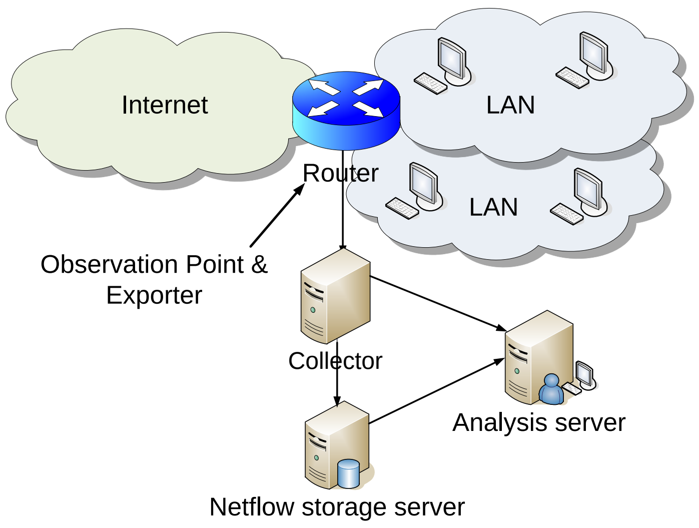


Figure 2.3: A typical NetFlow data-collection scenario.

- The ingress interface
- The source IP address
- The destination IP address
- The network layer protocol (e.g., IP)
- The source port (if UDP or TCP is used)
- The destination port (if UDP, TCP) is used or a code for ICMP
- The type-of-service information from the IP packet
- The duration of the connection
- The number of bytes sent from the source and from the destination

Whereas NetFlow v5 is a static data format, the recent version 9 is based on a modular structure with varying datatypes and very flexible usage. It builds the foundation for the IETF's specification of Internet Protocol Flow Information Export (IPFIX) [14, 59, 74], which is a universal flow information export standard.

A typical NetFlow collection scenario is exemplified in Figure 2.3: The IP traffic is observed at the *observation point*, which is typically the edge router of the system, by the *metering process* (also IPFIX nomenclature). Via an *exporter* the aggregated data is sent to the data *collector*. It is important to note, that the aggregation of flow information requires maintaining state for each open connection to finally report flow statistics on connection closure.

From a security researcher's perspective, the *ISPNetFlow* dataset has the slight drawback that there is no ground truth information on infections due to the lack of the underlying, full traffic capture. Therefore, no full content inspection is possible. Nevertheless, as will be shown throughout this thesis, the statistical data allows comparison of IP addresses to known malware IP blacklists and judgement of the usability of the – to be presented – BotFinder approach for the daily operation of large networks.

Chapter 3

Network Traffic Analysis

The characterization and classification of network traffic in a packet payload agnostic way – as required for this thesis's goal of content agnostic malware detection in networks – is inherently a hard task as only minimal information is available. Such information must be obtained from packet header observation and connection properties only. This chapter details how this information can be grouped into chronological structures called *traces*, which itself enable the extraction of statistical features.

An interesting feature, for example, is the assumed periodicity of malware communication with its *C&C* server. Stone-Gross et. al [71] support this assumption of periodic *C&C* traffic as they observed two main communication intervals for the Torpig botnet under their control: A fixed twenty minute interval for the upload of stolen data and a second, two hour long interval between updates of server information. As statistical features that can be attributed to typical bot behavior allow statistical analysis, this chapter investigates distinguishing properties between network traffic exhibited by malware and normal, benign, traffic in detail.

3.1 Available Information

Without Deep Packet Inspection (DPI), only packet header information is available. However, the headers allow reconstruction of transport layer connections to a representation similar to NetFlow. Therefore, for each flow,[1], the

[1]The words flow or connection are used interchangeably throughout this thesis.

following information is obtained:

- The IP addresses of the source and destination host,
- the ports at the source and destination,
- the number of bytes transferred to the source/destination,
- the number of packets transferred to the source/destination,
- the duration of the communication, and
- the start and end time of the connection.

All further analysis has to be based on this basic information.

3.2 Traces - Chronologically Ordered Connection Sequences

Over a longer network traffic observation period, all flows between two hosts A and B belong to the same "service" can be grouped together to build a basis for further analysis. As the simplest mean of distinguishing a service, one can define the property to share the same destination IP address, destination port and to use the same transport layer protocol. Therefore, the definition to group flows together as belonging to the same service – whereby a *C&C* server of a malware is explicitly considered as a service – is:

Two flows are considered to belong to the same service, when they have the same source IP address, destination IP address, destination port and transport layer protocol identifier.

Using the flow start time information to chronologically order the set forms a sequence of flows that is – in the following – referred to as *trace*, denoted $\mathcal{T}$. Such traces reflect the communication behavior of a two hosts with each other over a given timespan. If many connections are aggregated in a single trace, statistical properties of that communication can be extracted.

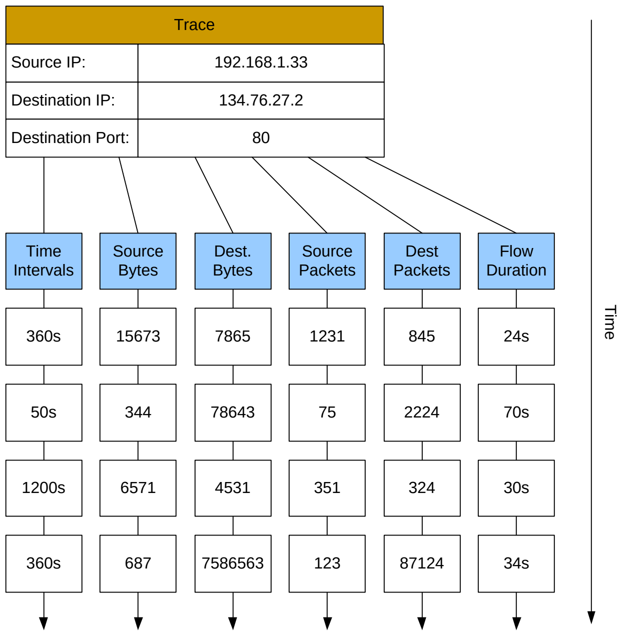


Figure 3.1: Example of the accumulated information in a trace.

3.3 Dimensions of Traces

Although the connection elements in a trace are ordered by start times, a trace posses multiple *dimensions* or vectors reflecting the available information from each flow in the trace as depicted in Figure 3.1: A timing dimension that contains the different start times (or intervals between start times), a byte transfer dimension for each communication direction, a packet count dimension for each direction, and the duration of each flow. The source IP address, destination IP address, and destination port (and other scalar information) complements the trace.

3.4 Basic Features of Traces

For traffic characterization it is relevant to identify statistical properties that allow trace distinguishing. In this thesis, the statistical *features* of a trace should especially capture typical malware behavior best. In this context, please

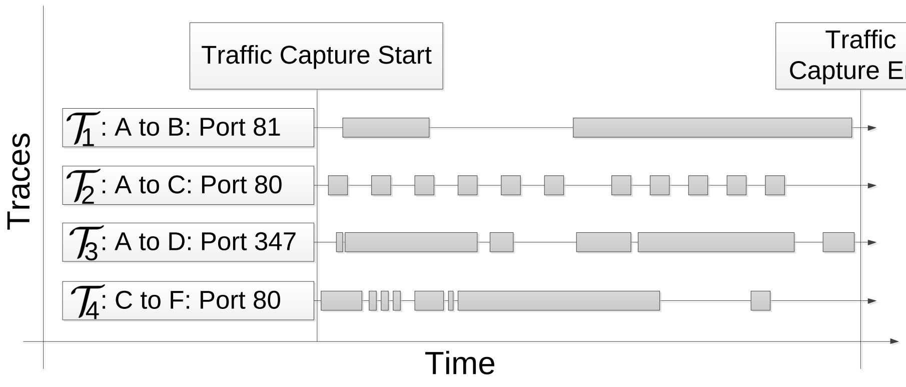

Traffic Capture Start
Traffic Capture End
$\mathcal{T}_1$: A to B: Port 81
$\mathcal{T}_2$: A to C: Port 80
$\mathcal{T}_3$: A to D: Port 347
$\mathcal{T}_4$: C to F: Port 80
Traces
Time

Figure 3.2: Traces with different statistical behaviors (in time and duration space).

note the difference between a trace dimension and a feature: A dimension is the vector of observed properties of the traffic such as the start time, byte transfer or connection duration. The concept of a feature describes a mathematical property such as “the average interval between connections” derived from these vectors. Although, in the following, only simple mathematical properties are used to calculate the basic features of a trace, there is no general limitation to the complexity or variety of potential features extracted. This especially covers behavioral models on traces or statistically inhomogeneous distributions.

Figure 3.2 exemplifies different shapes of traces. Here, the trace $\mathcal{T}_2$ from A to C on port 80 shows a highly regular behavior. In this example, the roughly constant distance between two flows – having a high periodicity – and the similar duration of communication allows for an accurate description of the whole trace by using the average time distance between flows and their average duration only.

In the following, basic features to efficiently describe malware traffic traces are presented. Please note that no feature is based on the *packet* dimension of the trace. This is because a strong correlation with the transferred bytes can be expected and the number of packets is heavily influenced by network setup properties like the Maximum Transmission Unit (MTU). Such properties do not belong to the network behavior of potential bots and the packet dimension is ignored in the remainder of the thesis.

3.4.1 Average Time Interval

The first feature considered is the average *time interval* between the start times of two subsequent flows in the trace. The botmaster has to ensure that all bots under her/his control receive new commands and update frequently. Communication from the *C&C* server to the bots following a push model is often impossible, as many infected hosts in private networks are behind Network Address Translation (NAT) boxes or not registered with the *C&C* server yet. A core assumption is that most bots use a constant time interval between *C&C* connections or a randomized value within an certain underlying, specific time interval. This leads to detectable periodicity in the communication. For the communication pattern, the botmaster has to balance the scalability and agility of his botnet with the increasing risk of detection associated with an increasing number of *C&C* server connections. Some bot variants already open random, benign connections [22, 21, 57] to distract signature generation (e.g., Li et al. [46]) and malware detection systems. Other approaches, such as "connect each day at time X" also suffer from issues like the requirement of synchronization between the bots' host clocks. Nevertheless, malware authors might craft their bots explicitly to not show periodic behavior. The potential impact of such randomization efforts on detection systems that exploit regularity are discussed in Chapter 8.

3.4.2 Average Duration of Connections

The *average duration* of communication between the bot and the *C&C* server is expected to be relatively constant, as bots often do not receive new commands and, as a consequence, most of the communication consists of a simple handshake: The bot requests new commands and the *C&C* server responds that no new commands have been issued. Thus, it can be expected that *C&C* traces behave in a similar manner. Nevertheless, the duration of communication is also dependent on the end host bandwidth of the client (bot) and inside a single trace, such timing durations remain constant. However, different network environments may impact the average duration feature and the same malware with identical traffic might create slightly different average durations. Nevertheless, such fluctuations may be minimal enough to still allow a positive indication of a bot infection. This argument is supported for the – to be presented – BotFinder system by a feature contribution analysis in Section 7.5.

3.4.3 Average Number of Source and Destination Bytes Transmitted

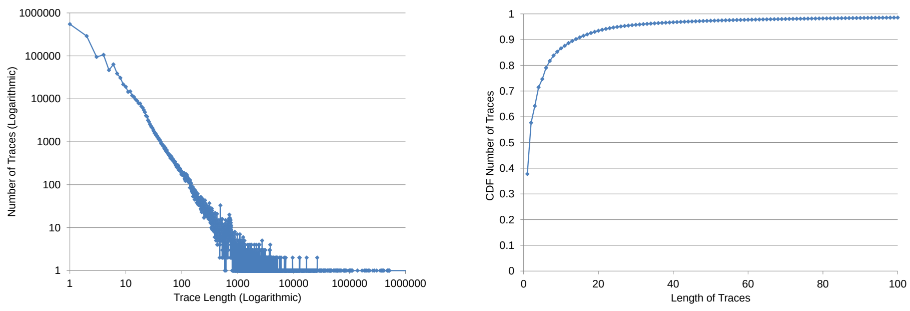
The *average number of source bytes* and *destination bytes* per flow is, similar to the duration of the connection, a recurring element in the bot's *C&C* communication. By splitting up the two directions of communication using source and destination bytes, it is possible to separate the request channel from the actual command transmission. That is, the request for an updated spam address list might always be of identical size, whereas the data transferred from the *C&C* server, containing the actual list, varies. As a result, a *C&C* trace is expected to contain many flows with the same number of source bytes. Similar considerations apply to the destination bytes – for example, when the response from the *C&C* server has a fixed format.

3.5 Malware Traffic

To collect traffic of malware samples, the Ant system (see Section 2.5) was used. On average 30 samples of Banbra, Bifrose, Blackenergy, Dedler, Pushdo and Sasfis, as presented in Section 2.4, were executed in a Windows XP VM for one to two days and all network traffic was recorded. To better separate normal network traffic events from regular *C&C* traffic, some of the following optimizations are applied.

3.5.1 Minimal Trace Length

As for every statistical property, the expressive power of a given number for real world properties grows with an increasing number of observation or experimental repetitions to obtain the number. Analog, each trace feature requires a minimal amount of flows in the trace to derive a meaningful statistical interpretation. For example, the statistical distance between just two individual points is of nearly zero expressive power to describe a statistical behavior, whereas a dataset of 100 collected intervals already allows a quantitative description. Consequently, only traces of a certain length greater than $|\mathcal{T}|_{min}$ are considered for feature extraction. The selection of this threshold is hard as it has to be high enough to drop infrequent and arbitrary requests (which are not of interest for *C&C* traffic detection) but low enough to capture recurring *C&C* connections with respect to the overall observation time. The general fact, that



(a) Overall traces (b) One-Day traces

Figure 3.3: Trace length distribution.

such a minimal threshold exists is consistent with the fact that command and control traffic consists of multiple connections between the infected host and the *C&C* server.

If one investigates normal, benign traffic as assumed in the *LabCapture* dataset, a large number of short traces and a fast decreasing number of longer traces is observed. As illustrated in the double logarithmic plot in Figure 3.3(a), the vast majority of traces is of very short length, only 2.7% are longer than 50 flows. The Cumulative Distribution Function (CDF) shown in Figure 3.3(b) highlights this behavior: although only trace lengths with less than hundred datapoints are shown, the CDF quickly reaches 98.5%. Moreover, 37% of all traces in the *LabCapture* dataset are of length one, the CDF up to length five already covers 75% of all traces.

Given the periodicity information of 20-minute-intervals from Stone-Gross et. al [71] for the Torpig botnet, a limitation to an initial minimal trace length of $|\mathcal{T}|_{min} = 50$ seems reasonable given one collects traces for one or a few days. By this number, the trace analysis workload is reduced to 2.5% and the statistical quality of the trace should allow estimations of periodic behavior.

However, for the analysis of *C&C* communication in a controlled environment, a shorter $|\mathcal{T}|_{min}$ may be chosen if manual inspection reveals that the short traces do not negatively impact the overall trace quality. The impact of minimal trace length selection for an automated processing of the malware sets of this thesis is presented in Section 6.2.

3.5.2 Identification of *C&C* Traffic – Purification

The identification of relevant *C&C* traffic in network traces is of importance as it cleans the input data from traces that have been falsely attributed to bot communication. The problem of *C&C* traffic classification is especially hard if fully automated approaches, for example, for automated machine learning, are used.

Any classification mechanism has to reduce the uncertainty which malware-generated traces are meaningful command and control interactions and which traces are just random, additional "noise". Such noise might either be intentionally generated by the bot under investigation [22, 21, 57], by the operating system itself – for example update services, network discoveries etc. – or other applications running on the machine. The introduction of benign traffic is an advanced method to cloak the malware's own *C&C* communication and to counter automatic signature generation systems.

For this *purification* of training input traces, the traces are classified into three different groups reflecting the attribution to the malware sample: The first group contains all traces that are classified to a different service or are considered non-malicious with a very high probability. Typical traces in this set are whitelisted connections such as to internal servers, common Internet services such as Microsoft Update, or other requests that can be attributed to well known and documented benign services. The second group contains malicious traces that have a high probability to belong to a bot's communication. Classification to this set is realized by using, for example, one of the following traffic identification methods:

1. A manual way is to leverage third party knowledge and perform traffic inspection – if the traffic is unencrypted – to compare the packet payloads to known signatures or special communication patterns.

2. Another option is the comparison of the destination IPs of flows to a list of known *C&C* servers, which is an easy to automate and efficient task.

3. A more advanced and automated technique that allows identification of previously unknown *C&C* servers is JACKSTRAWS [36] by Jacob et al., an approach that leverages additional system call information from the bot sample's execution.

The method selection process highly depends on the final application of the traces. The third group of traces is formed from the traces that are not proven to be malicious but that are exhibited by the malware, which might have not shown traffic that matches the previous two groups.

Purification Strictness

One can distinguish between two different modes of trace purification: In the first configuration – (*standard*) – the purification is performed for each sample and if a network trace matches a blacklisted IP, only this trace (and other matching traces, if any) of the sample is(are) used for further investigation. If samples do not contain any traces whose destination IP addresses match a known, blacklisted, IP address, all traces are considered matching. The second mode – (*strict*) – ignores samples that do not contain matching traces and only accepts traces from the set of verified malicious traces.

3.6 Comparison of Malware Traffic with Benign Network Traffic

The core element to detect malicious traffic in network traces without the use of packet payloads are recurring statistical properties of the traffic. Such properties enable methods to distinguish malicious traffic from benign traffic and, as aforementioned, the main assumption followed in this thesis is that the distinguishing feature of bot traffic is a more periodic behavior than typical benign traffic.

To investigate the periodicity in traces, the feature average $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$, with feature values $x_i, i \in [0, N]$ for N flows in the trace, and it's standard deviation $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$ is calculated. σ is a statistical measure of the the variation of data around the average and therefore captures the periodicity in communication behavior. If a bot connects frequently in similar time intervals to the *C&C* server, the average will capture this interval and the standard deviation will be low if the spread or dispersion of data is low. As a measure of periodicity, the *relative standard deviation* is used, which is defined as σ/μ. A low relative standard deviation therefore expresses a high periodicity of the traffic feature. Please note that the average and relative standard deviation is calculated on a per-trace level. Low values therefore *do not* indicate that all traces behave the same, but that each trace in itself is relatively periodic.

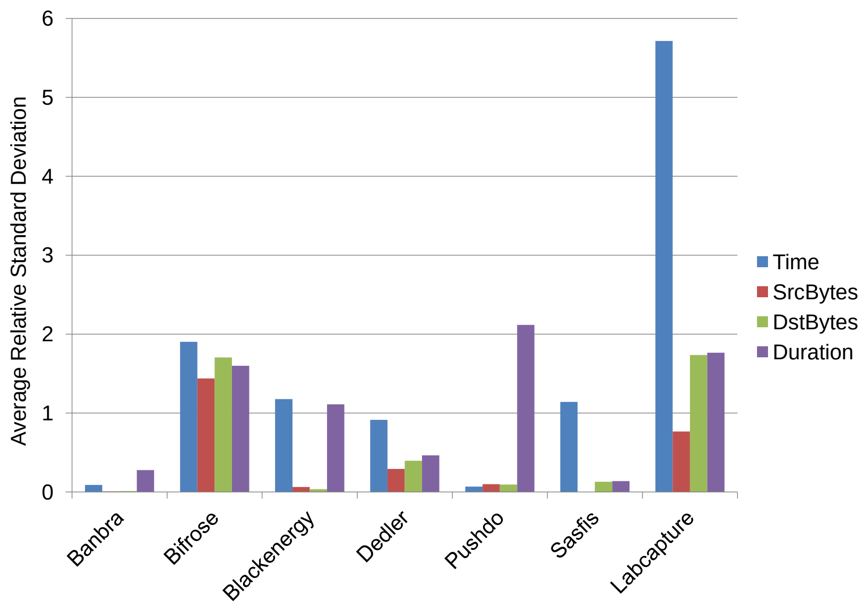


Figure 3.4: Feature relative standard deviations of malware and benign traffic samples: The more periodic a feature, the lower the bar.

For each malware family, the averages over all relative standard deviations of a given feature are computed, which yields the values depicted in Figure 3.4. Here, the difference between malware and benign traffic becomes highlighted as the relative standard deviations for the "normal" traffic in the *LabCapture* dataset are high compared to the bot's traffic. Especially the Banbra bot family has very periodic traffic (low average relative standard deviation), whereas Bifrose's traffic is significantly more random. It is also interesting to see, that the different dimensions analyzed in this experiment are not necessarily correlated. For example, the Pushdo traces show very high periodicity for time and the average number of bytes transferred to the source and destination, but the duration of connection highly fluctuates. In some dimensions, bots like Bifrose are even more non-periodic than normal traffic (e.g., in the source bytes dimension). Nevertheless, as a first result of this comparison one can state that the bots under investigation on average show a significantly higher level of periodicity than normal user traffic.

Chapter 4

BotFinder Design

In this Chapter, the high periodicity of malware traffic is exploited to create BotFinder, a system that detects malware infections in network traffic by comparing statistical features of the traffic to previously-observed bot activity.

BotFinder operates in two phases: a *training* phase and a *detection* phase. During the training phase, the system learns the statistical properties that are characteristic of the command and control traffic of different bot families. Then, BotFinder uses these statistical properties to create models that can identify similar traffic. In the detection phase, the models are applied to the traffic under investigation. This allows BotFinder to identify potential bot infections in the network, even when the bots use encrypted *C&C* communication.

Figure 4.1 depicts the various steps involved in both phases: First, input for the system is obtained. In the training phase, this input is generally generated by executing malware samples in a controlled environment such as Anubis [5], BitBlaze [67], CWSandbox [78], or Ether[19] and by capturing the traffic that these samples produce. As described in Section 3.5, throughout this thesis the training input is obtained by using the Ant system that utilizes Anubis binaries. In the second step, the flows in the captured traffic are reassembled; a step that can be omitted when NetFlow data is used instead of full packet captures. In the third step, the flows are aggregated in traces as described in Section 3.2 – chronologically-ordered sequences of connections between two IP addresses on a given destination port. BotFinder then extracts five statistical features for each trace in the fourth step. These statistical features are the already introduced features of average *time* between two subsequent flows

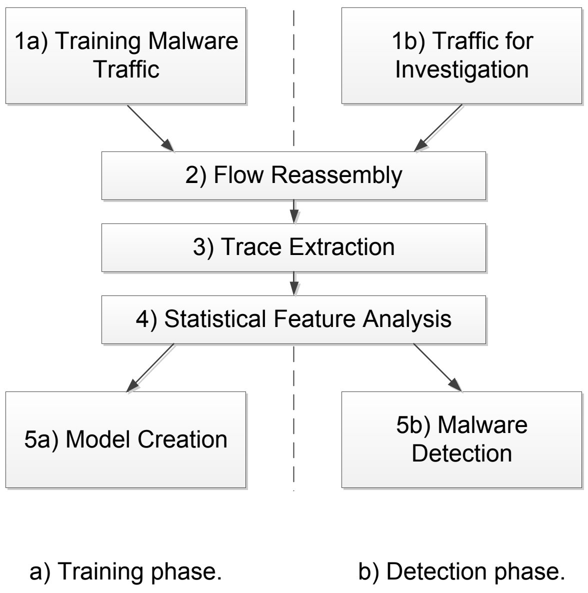


Figure 4.1: General architecture of BotFinder. During the training phase (a), malware samples are run and models are created based on the statistical features of the bots' network behavior. During detection (b), BotFinder analyzes NetFlow or full traffic captures and compares the extracted network features to the established models.

in the trace, the average *duration* of a connection, the *number of bytes* on average transferred to the source, and the number of bytes on average transferred to the destination. Additionally, a *Fourier Transform over the flow start times* in the trace is calculated. This Fast Fourier Transform (FFT) allows to identify underlying frequencies of communication that might not be captured using simple averages. Finally, in the fifth step, BotFinder leverages the five features to build models. During model creation, BotFinder clusters the observed feature values. Each feature is treated separately to reflect the fact that not always correlations between features are observed: For example, a malware family might exhibit similar periodicity between their *C&C* communications, but each connection transmits a very different number of bytes. The combination of multiple clusters for each of a bot's features produces the final malware family model.

When BotFinder works in the detection phase, it operates on network traffic and uses the previously-created models for malware detection.

It is important to note that BotFinder does not rely on any payload information of the traffic for the whole process, but works on the statistical properties exhibited by the *C&C* communication only.

In the following, the steps involved are explained in greater detail.

4.1 Input Data Processing

The input to BotFinder is either a traffic capture or NetFlow data, which is a dominant industry standard for traffic monitoring and IP traffic collection. During the training phase, malware samples are executed in a controlled environment (as done in the Ant environment), and all network traffic is recorded. In this step, it is important to correctly classify the malware samples so that different samples of the same malware family are analyzed together as described in Section 2.4. Of course, incorrectly classified samples are possible and might affect the quality of the produced models. However, as explained later in Section 4.5.4, BotFinder tolerates a certain amount of noise in the training data.

4.2 Flow Reassembly

In this step, flows are reassembled from captured packet data. A flow is identified by the 5-tuple of source IP address, destination IP address, source port, destination port, and transport protocol ID (UDP or TCP). For each connection, properties such as start and end times, the number of bytes transferred in total, and the number of packets are extracted. As a final result of this reassembly step, BotFinder yields aggregated data similar to NetFlow. For all further processing steps, the system only operates on these aggregated, content-agnostic data. If NetFlow data is available, BotFinder directly imports it.

4.3 Trace Extraction

A core element in BotFinder is the concept of *traces* as described in Section 3.2: A trace $\mathcal{T}$ is a sequence of chronologically-ordered flows between two network endpoints that share the 4-tuple source IP, destination IP, destination port and protocol ID. Effectively, the source port information from the previous step is dropped as recurring connections may be created on random different source ports without changing any of the relevant connection properties.

For the trace extraction as many connections as possible have to be collected and ordered along the aforementioned 4-tuple. The trace captures the mid to long term behavior of communication between two hosts on a given service.

4.4 Feature Extraction

After trace generation, BotFinder processes each trace to extract relevant statistical features for subsequent trace classification. Inherently, the feature extraction is the systematical core of BotFinder as it defines all properties used for the following model creation and finally the malware detection in network traffic. In general, the four features introduced in Section 3.4 of average time interval, average number of source bytes, average number of destination bytes, and the average duration of connections are calculated.

In order to detect underlying communication regularities, a FFT is calculated over the *C&C* communication. In this step, the trace is sampled like a binary signal by assigning it to be 1 at each connection start, and 0 in-between connections.

The general purpose of a Fourier transform is the trigonometric approximation of an aperiodic signal to a function composed of cosine or sine waves. In its continues form and using Euler's formula ($e^{ix} = \cos x + i \sin x$) to simplify the writing by using complex exponents, a Fourier transform of a function $f(t)$ can be expressed by

$$\mathcal{F}[f](\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(t)\, e^{-\mathrm{i}\omega\cdot t}\, \mathrm{d}t \tag{4.1}$$

with ω being the angular frequency $\omega = 2\pi/T$ with the period T. However, as BotFinder obtains and samples the flow information in discrete time intervals, a Discrete Fourier Transform (DFT) needs to be used. Let $a = (a_0, ..., a_{N-1})$ be the series of N sampled binary datapoints that represent the trace flows. Analog to Equation 4.1, the complex fourier coefficients $\hat{a}$ for the DFT can be expressed as:

$$\hat{a}_k = \sum_{j=0}^{N-1} e^{-2\pi \mathrm{i} \cdot \frac{jk}{N}} \cdot a_j \tag{4.2}$$

Please note that the prefactor $\frac{1}{\sqrt{2\pi}}$ in Equation 4.1 normed the infinitesimal summation over the unit circle whereas in Equation 4.2, the sum is calculated over all elements of the series a. The inverse DFT has to be normed accordingly:

$$a_k = \frac{1}{N} \sum_{j=0}^{N-1} e^{2\pi \mathrm{i} \cdot \frac{jk}{N}} \cdot \hat{a}_j \tag{4.3}$$

The mathematical description in Equation 4.2 becomes clear if one represents the discrete values a as values $a_k = A(z_k)$ of the polynomial

$$A(z) = \tfrac{1}{N} \left(\hat{a}_0 + \hat{a}_1 z + \hat{a}_2 z^2 + \cdots + \hat{a}_{N-1} z^{N-1} \right) \tag{4.4}$$

with complex coefficients $\hat{a}_0, ..., \hat{a}_{N-1}$. The arguments $z_0, z_1, \ldots, z_{N-1}$ are chosen homogenously from the unit circle:

$$z_k = e^{\frac{2\pi \mathrm{i}}{N} k} = \cos(\tfrac{2\pi}{N} k) + \mathrm{i}\ \sin(\tfrac{2\pi}{N} k) \tag{4.5}$$

The key to carry out the transform to frequency space is to express z_k via a time based function that circles on the unit circle

$$z(t) = e^{2\pi \mathrm{i} \frac{t - t_0}{T}}. \tag{4.6}$$

At times $t_k = t_0 + \frac{k}{N} T$, this function returns 1, which finally resolves to a_k using Equation 4.2. The exponents of $z(t)$ yield

$$z(t)^k = e^{k \cdot 2\pi \mathrm{i} \frac{t - t_0}{T}} = \cos(2\pi k \tfrac{t - t_0}{T}) + \mathrm{i}\ \sin(2\pi k \tfrac{t - t_0}{T}), \tag{4.7}$$

which is periodic in T/k. As a consequence, the measured series a can be described by a constant value at $k = 0$, a fundamental frequency at $k = 1$ and superpositions of frequencies at $k > 1$.

If a consists of real values only, the DFT spectrum is composed of $N/2$ values mirrored at the folding frequency at $k = N/2$. The term FFT is often used synonymous with DFT, although it classifies a set of algorithms that allow a faster calculation of the DFT than a plain implementation with $\mathcal{O}(N^2)$. Most algorithms such as Cooley and Tukey [17] use a Radix-2-FFT approach that divides and conquers the DFT calculation by splitting the series into sub-series and calculating each DFT separately. Hereby, the DFT calculation is improved to $\mathcal{O}(N log_2 N)$. As the divide step requires to split the data series, the algorithms work optimal for series that have a length of a factor of two: $N = 2 \cdot M$. If the series is not highly composite, the transform algorithm is unable to divide the series efficiently and the calculation is slow compared to a highly composite series.

To calculate a high-quality FFT, BotFinder samples the signal with a frequency of two times the *Nyquist* frequency [56], which is the minimal frequency that allows to distinguish between individual data points. This implies a sampling interval of 1/4th of the smallest time interval in the trace and ensures that the trace is not "undersampled". However, if the distance between two flows is extremely small and large gaps occur between other flows of the trace, this sampling method can lead to a significant amount of data points. In such cases, the length of the FFT is limited to $2^{16} = 65,536$ datapoints and minor undersampling is accepted. This value was chosen as the FFT is fastest for a length of the power of two, and, with this value, only few datapoints are (under)sampled in a way that two flows appear as one. More precisely, for the *C&C* traces observed in this thesis, 18% showed undersampling, which resulted in a median of only 1% of the start times that were sampled together.

In the next step, the Spectral Density (SD) of the transform is calculated over the sampled trace and the most significant frequencies are extracted. The FFT effectively returns the complex fourier coefficients for frequencies between 0 (the constant) and frequencies up to half of the sampling frequency. To obtain the spectral density, the square of the magnitude of each frequency f_k has to be calculated by $magnitude(f_k)^2 = \mathrm{Re}(a_k)^2 + \mathrm{Im}(a_k)^2$. Picking the frequencies with the highest magnitude from the spectrum reveals the most significant frequencies. These FFT peaks correlate with time periodicities and are resistant against irregular large gaps in the trace (as will be shown in Chapter 8). When malware authors randomly vary the *C&C* connection frequency within a certain window, it will lower the FFT peak. However, despite the randomization the FFT peak remains detectable at the underlying frequency and allows the detection of the malware communication.

4.5 Model Creation via Clustering

Models are created using clustering on previously-extracted features such as average time, average duration, average source bytes, average destination bytes, and FFT frequencies. Please note that other features can easily be integrated, as long as they are comparable by Euclidian distances.

The clustering process is a hierarchical approach that performs multiple grouping steps before actually generating the clusters. The starting point of the analysis is an initial grouping of all traces that belong to the same malware family. For the bots under investigation in this thesis, those are the six sets for Banbra, Bifrose, Blackenergy, Dedler, Pushdo and Sasfis. In the next step, each feature is clustered individually in each set because traces with, for example, the same connection duration exhibit completely different average time intervals between each flow. To not artificially introduce a correlation that does not exist in reality, all features are clustered separately.

After clustering, typically a number of rather large clusters is observed that can be assumed to contain the actual malware-specific behavior. Furthermore, some smaller clusters with more diverse data (lower clustering quality) are seen, and even individual traces that might be considered as false attributions to the malware sample can be observed. As a consequence, very small clusters are dropped, which makes BotFinder robust against diverse and noisy traffic. If different training malware samples exhibit different noisy traces, these traces will be clustered either to a very large and loose which is not relevant through the quality measure or to a cluster with only one element.

The final model $\mathcal{M}$ itself spans the five features, each containing a collection of cluster centers. In human terms, a model can be understood as:

An average interval between connections of 2,100 seconds, a transfer of 51kB to the source, 140 bytes to the destination, a flow duration of 10 seconds, and a communication frequency of around 0.04Hz indicate a Dedler infection.

4.5.1 The CLUES Clustering Algorithm

To cluster the trace-features for a bot family, the CLUES (CLUstEring based on local Shrinking) clustering algorithm [77] is used. This algorithm allows non-parametric clustering without an initial determination of the expected number of clusters.

CLUES iteratively applies three procedures:

1. Shrinking: Local shrinking is based on the rationale of gravitational clustering [79, 42], which treats all data points as unit mass and zero velocity particles and applies an arbitrary gravitational field between the data points. Denser populations attract data points from the sparsely populated areas and, over a number of iterations, the data points converge to so-called focal points. In CLUES, this step is realized using a mean-shift algorithm [26, 15], but governed by the K-nearest neighbor approach [47] instead of kernel functions.

2. Partition: After the shrinking process, the calibrated set of data points is used to obtain the actual cluster formations or, in other words the membership function.

3. Determination of K, the optimal number of clusters: In each step that K is gradually increased, a cluster strength measure index function is calculated. For this index funcation either the CH index by Calinski and Harabasz [10] or the Silhouette index by Kaufman and Rousseuw [39] is used. K is chosen to give the optimum of the selected index function. In our analysis, we used the Silhouette index following the slightly better results for this indexing method found by Wang et al. [77].

As CLUES is computationally challenging and relatively slow for large datasets due to the iterative processes involved (for details see [77]), the authors introduced a speed factor α that allows to balance between clustering quality and speed. A larger α delivers more accurate clustering but significantly increases the run time of the algorithm. In its recommended default setting [11] α is set to 0.05 and, additionally, the number of the allowed maximum of iterations is defaulted to 20.

Fortunately, the model clustering datasets originating from malware samples are typically in the range of tenth to a few hundred datapoints and consist of plain double values comparable using the most simple Euclidian distance. This allows to balance the algorithm more towards clustering accuracy by using $\alpha = 0.20$ and to set the maximum number of iterations to 200. The latter excludes clustering limitations raised by a too limited number of iterations. However, typically the algorithm converges in five to ten iterations.

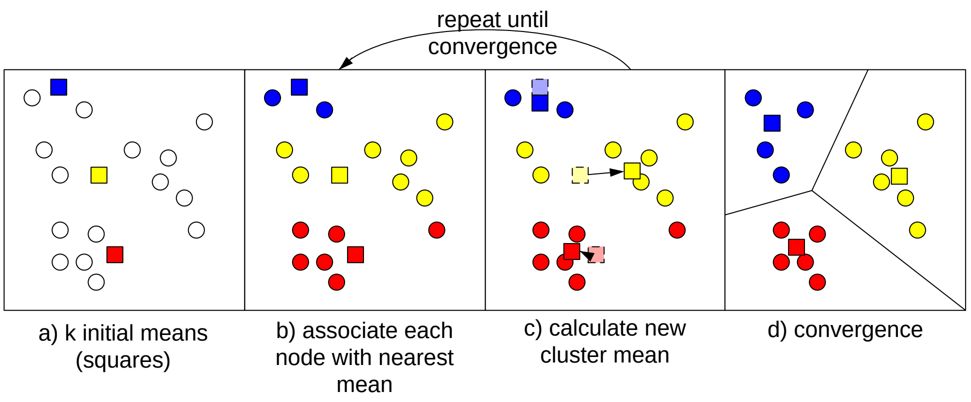

repeat until convergence
a) k initial means (squares)
b) associate each node with nearest mean
c) calculate new cluster mean
d) convergence

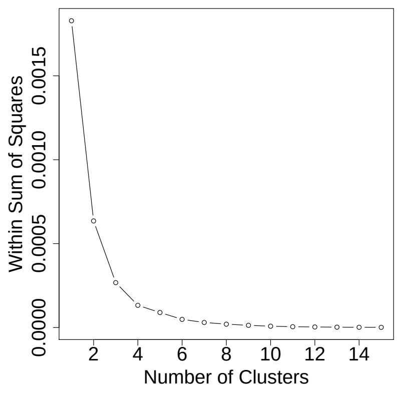
Figure 4.2: Illustration of the k-means clustering algorithm.

4.5.2 CLUES Compared to k-means

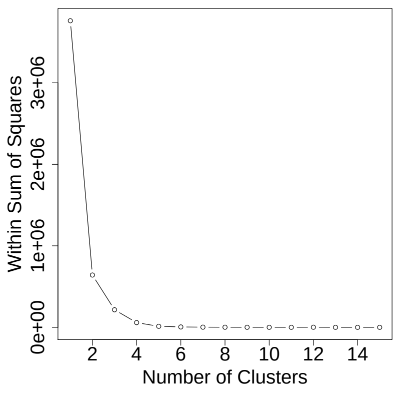
Although Wang et al. demonstrated impressive results clustering four typical datasets [77], the applicability of CLUES was verified for the specific structure of malware datasets by comparing it to the well-known k-means algorithm [48, 34]. This simple algorithm incrementally optimizes a pre-specified number of k clusters. As depicted in Figure 4.2, each center is initially chosen randomly and the remaining data points are assigned towards the closest cluster center [Step a) and b)]. In each iteration, the cluster mean is calculated and used as the new cluster center [step c)]. Repeatedly, the assignment is performed again and the cluster centers are recalculated. As a consequence, the average distance of each point to its centroid decreases until final convergence is reached.

Effectively, each iteration minimizes the sum over the cluster's within sum of squares (wss) $\sum_{j=1}^{k} wss = \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^j - c_j||^2$ of n datapoints that group to k clusters with centroids (means) $c_{1..k}$. This measure is in general used to rate the overall success of k-means and to compare results for different k. As the clustering quality also depends on the initial random position of the cluster centers, the algorithm is typically run from $n = 10$ to $n = 50$ times and the best fit – the overall minimal sum of wss – is chosen. Please note, that each result only represents a local minimum, which does not necessarily reflects the global minimum.

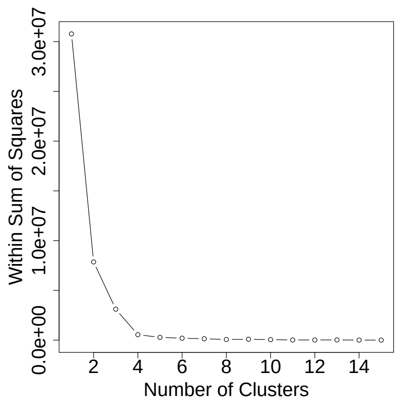
To find a good k for comparison to CLUES, k-means was run 50 times for all $k \in [1, 15]$ and the the sum of wss over k was plotted as shown in Figure 4.3. In



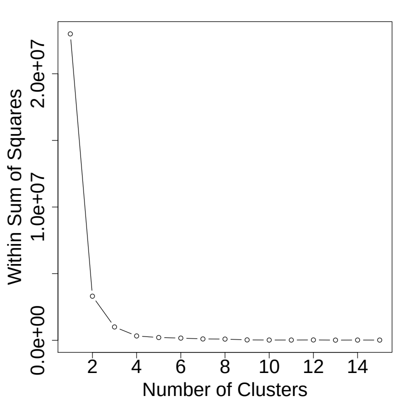
(a) Banbra



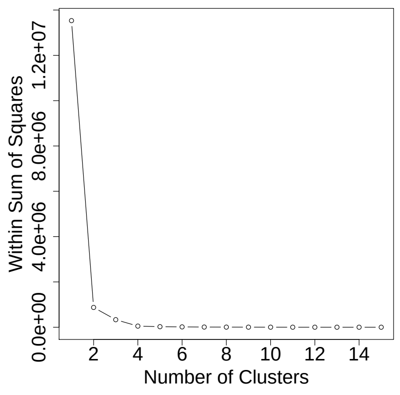
(b) Bifrose



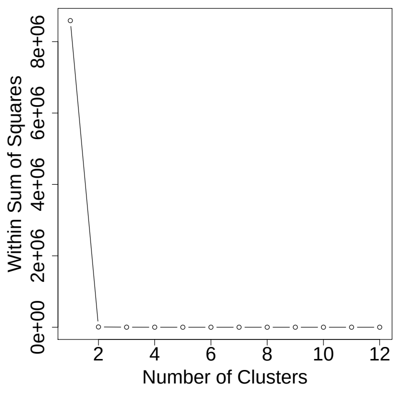
(c) Blackenergy



(d) Dedler



(e) Pushdo



(f) Sasfis

Figure 4.3: Within sum of squares of k cluster for the average time feature.

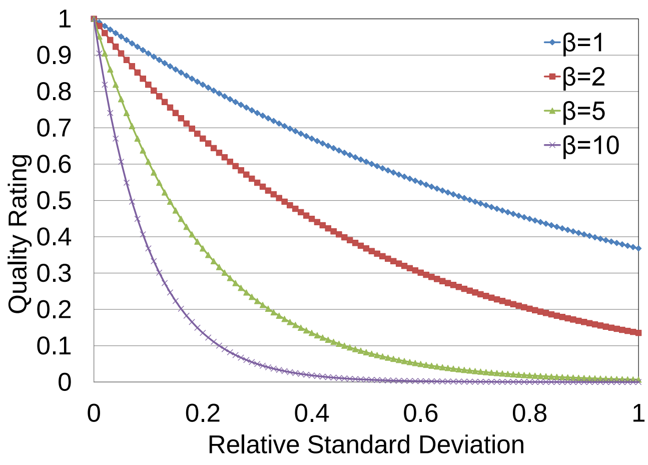


Figure 4.4: Quality rating function by relative standard deviation.

the next step, the selection of an appropriate k was performed either manually or in an automated way: During manual selection, a k was chosen that allowed a relatively low sum of wss balanced with the number of clusters. In a second step, this procedure was automated to select k in an unsupervised way, so that adding a cluster $k + 1$ would lower the sum of wss only by a given fraction, typically set to 0.5. In other words, adding a cluster should lower the wss at least by 50%. Both methods generate similar, often identical clustering results as the fully automated, non-supervised CLUES algorithm. In certain cases, even slightly better cluster formation than with k-means was found. The same holds for the manually supervised selection. Therefore, CLUES is considered to be an ideal fit for the specific clustering scenario faced by BotFinder.

4.5.3 The Quality Rating Function

As aforementioned, the cluster quality is typically measured using the within sum of squares or other, more advanced methods as the CH or Silhouette index. For BotFinder, the quality is expressed using a dedicated exponential quality rating function that is based on the within sum of squares. Hereby, the standard deviation of each cluster is calculated based on the sum of squares SS. Connected via the variance $V = SS/(N-1)$ with N being the cluster size (number of elements in the cluster), the standard deviation σ follows to be:

$$\sigma = \sqrt{\frac{SS}{N-1}}. \tag{4.8}$$

As in the periodicity rating in Section 3.6, the relative standard deviation $rsd = \sigma/c$ with c being the cluster average is used. This rsd is finally used as an input to an exponential cluster quality rating function, which is illustrated for four β values in Figure 4.4:

$$q_{cluster} = \exp^{-\beta \cdot rsd} \tag{4.9}$$

This quality measure reflects the uncertainties inherited by trace collection and feature extraction in the previous steps. A low quality rating represents a low trust in the capability of the rated cluster to describe a malware sample sufficiently. Consequently, the higher β, the faster the quality rating function decreases and the less trust is placed in clusters with increasing relative standard deviations.

4.5.4 The Final Model

The final models $\mathcal{M}$ consist of a hierarchical structure as depicted in Figure 4.5. The highest level builds the malware family $m \in \mathcal{M}$, which is itself divided into the different features under analysis. For each feature, a number of clusters represent the actual model. The cluster size (in traces) in relation to the total amount of traces reveals the relevance of the cluster. Too small clusters are dropped (currently, only clusters of size 1). The most relevant information is the cluster center (or centroid) and its standard deviation. Based on these information, the model quality is calculated.

4.6 Model Matching during Detection

During the detection phase, BotFinder matches traces from the traffic under investigation with the models $\mathcal{M}$ created during the training phase. Hereby, each statistical feature of a trace $\mathcal{T}$ is compared to the clusters generated for each model $m \in \mathcal{M}$.

The detailed process (including all optional conditions that will be defined in the following subsections) is shown in Algorithm 1: The trace $\mathcal{T}$ is compared

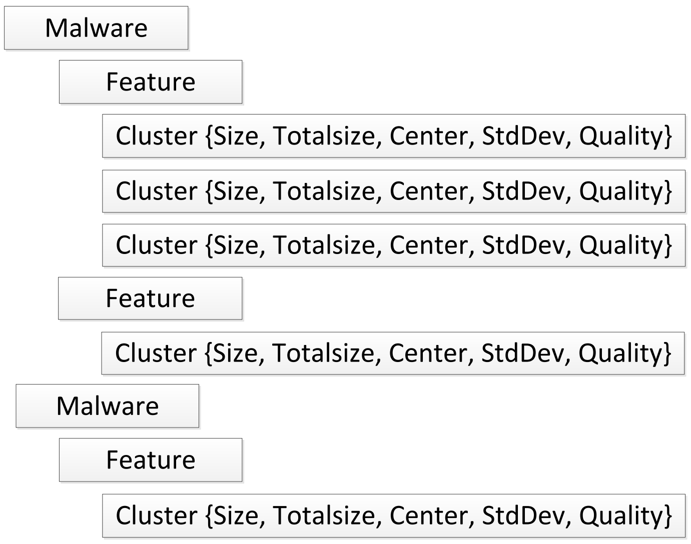

Malware
Feature
Cluster {Size, Totalsize, Center, StdDev, Quality}
Cluster {Size, Totalsize, Center, StdDev, Quality}
Cluster {Size, Totalsize, Center, StdDev, Quality}
Feature
Cluster {Size, Totalsize, Center, StdDev, Quality}
Malware
Feature
Cluster {Size, Totalsize, Center, StdDev, Quality}

Figure 4.5: Final model structure.

to all models (Line 6 to Line 20) and for each model m, a scoring variable γ_m is used. As shown in Line 6, all features are compared to the model individually and add to γ_m. The $match()$ function is detailed separately in Algorithm 2 and quantifies how well the cluster was hit in a range of $[0, 1]$.

If the matching is good enough – the scoring value is equal or larger than a predefined threshold α – the model is considered to be matched. The best matching model, which is the one with the highest γ_m, is finally returned as BotFinder's detection result (Line 30).

4.6.1 Requirement of Minimal Number of Hits

To reduce false positives and to not rely only on a single feature alone, BotFinder allows the user to optionally specify a minimal number of feature hits h. This means in addition to the requirement to match $\gamma > a$, the trace has to match in h features at least. This rule avoids accidental matches solely based on, for example, time periodicities. Setting $h = 3$ requires to not only match in two feature such as average time and FFT, but also in either the average duration or one of the byte transmission features. In Algorithm 1, this optional test is shown in lines 17 and following.

4.6.2 Increasing the Scoring Value

To match a trace to a cluster, the trace feature average x and its standard deviation σ_x, the cluster center c, the cluster's standard deviation σ_c, and the cluster's quality $qual_c$ is used.

BotFinder considers the value x to be matching the cluster with center c if it lies in the area of $x \in [c - 2\sigma, c + 2\sigma]$. The choice of two times the standard deviation is motivated by the assumption that matching traces may be gaussian distributed around the cluster average: For such a normal distribution, 95% of all values of the distribution are located in the range of $\pm 2\sigma$. In the context of malware detection quality this represents a 95% probability that a malware trace actually matches the cluster it belongs to.

Algorithm 2 details the trace-feature to cluster matching: If the trace matches – x is near to c – BotFinder either returns the quality of the matching cluster or performs additional error calculations: Hereby, the accuracy of the trace-feature that is matched to the cluster is considered as well: Input traces can be considered to be of a high statistical quality when the feature averages can be calculated with low standard deviations. To reduce false positives in the detection stage, this quality can be used to reduce the influence of low quality traces on the matching decision. Using $qual_c \cdot \exp(-\delta \cdot \sigma_x / x)$, both, the cluster quality and the trace quality, is used.

The presented calculations are mathematical representations of the statistical confidence placed in traces and clusters. Clusters consisting of loosely similar traces are considered less expressive than large clusters of tight and well clustering trace features. If a tight cluster is hit by a highly periodic trace, γ_m is increased most. If the same cluster is hit by a trace with low relative standard deviation for the feature, even the contribution of high quality clusters degrades exponentially with the rsd. Non-periodic traces that hit loose clusters barely increase γ_m.

Algorithm 1 Basic model matching algorithm

Input: a trace $\mathcal{T}$ to analyze and the set of models $\mathcal{M}$

Output: the detection scores for each model

```
1:
2:  $modelmatches = dict()$                          ▷ a dict is a simple key-value mapping
3:  for all models $m \in \mathcal{M}$ do                        ▷ loop through the models
4:      $\gamma_m = 0$
5:      $hits_m = 0$
6:      for all features $f \in \mathcal{F}$ do                  ▷ $f$ is e.g. “average duration”
7:          $t = \mathcal{T}[f]$                     ▷ the trace value and $\sigma$ for the feature $f$
8:          $cx = list()$                             ▷ stores the results for each cluster
9:          for all clusters $clus \in m_f$ do
10:             $cx.append(match(t, clus))$                        ▷ calculate the score
11:         end for
12:         if $max(cx) > 0$ then
13:             $hits_m = hits_m + 1$
14:         end if
15:         $\gamma_m = \gamma_m + max(cx)$                                  ▷ increase score
16:     end for
17:     if $hits_m > h$ then                                        ▷ optional check for h
18:         $modelmatched[m] = \gamma_m$
19:     end if
20: end for
21: $\gamma_{max} = 0.0$                                                ▷ find the best match
22: $m_{max} =$ “”
23: for all $m \in modelmatched$ do
24:     if $modelmatched[m] > \gamma_{max}$ then
25:         $\gamma_{max} = modelmatched[m]$
26:         $m_{max} = m$
27:     end if
28: end for
29: if $\gamma_{max} \geq \alpha$ then                                        ▷ hit good enough?
30:     return “Model matched:” $m_{max}$
31: else
32:     return “No match”
33: end if
```

Algorithm 2 Individual feature matching

Input: $x, \sigma_x, c, \sigma_c, qual_c$

Input: an *erroraware* flag as an option

Output: the matching score (to increase γ_m)

```
function MATCH($tracefeature, cluster$)
    if $x \geq c - 2 \cdot sd$ and $x \leq c + 2 \cdot d$ then
        if erroraware then
            return $qual_c \cdot \exp(-\delta \cdot \sigma_x / x)$
        else
            return $qual_c$
        end if
    else
        return 0
    end if
end function
```

Chapter 5

Deployment Considerations, Implementation, and Performance Benchmarking

This chapter details the implementation of BotFinder in Python and the targeted deployment scenarios. Moreover, a performance evaluation investigates the processing steps that require high computational efforts. Additionally, it highlights the overall processing speed, which is sufficient to handle large scale networks with millions of hosts in real-time.

5.1 Targeted Deployment Scenario

Depending on the network size, BotFinder is intended to be deployed in different locations in the network. In larger networks, system administrators typically already obtain network statistics information using Cisco's NetFlow or similar technologies, e.g., the novel IPFIX [14, 59, 74] standard based on NetFlow 9, to assess the network's service quality and proactively detect problems in the network. In such networks, the NetFlow information can be directly fed into a BotFinder analysis server as depicted in Figure 5.1. Here, the following definitions are used:

- **Edge routers** are routers deployed at the intersection between the inner ISP's network and a network under foreign control. Modern routers often support to automatically keep track of connections and log statistical information about involved IP-addresses, ports, connection durations and much more for later analysis.

- The **Data Collection Server** (or "collector") is a server that typically already exists in a larger network and that is used by the network administrator as a storage for network analysis. Such analysis may cover traffic distributions, traffic loads and in general network health monitoring. In the context of BotFinder, the server is just a data storage device, which holds the collected NetFlow/traffic information from the network.

- **Analysis Servers** are the core element of BotFinder. They read incoming traffic or NetFlow data and perform the trace assembly. As the trace generation requires large amounts of memory, BotFinder is able to run on multiple analysis servers in parallel that cover different segments of the network, e.g., different source IP address ranges. The analysis server also processes the traces in regular intervals and reports indications of bot infections to the network administrator.

- **Computational Support Servers** are processing units accepting a trace and performing the relevant computations for statistical analysis such as averaging and the Fast Fourier Transform.

Generally, the network's edge routers collect NetFlow information and submit the NetFlows to a central data collection server. Special considerations for BotFinder need to be taken if the NetFlow data is sampled, for example due to large traffic loads at the router: In such cases, the sampling rate – such as "every fifth connection is captured" – needs to be known and BotFinder needs to re-assemble the traces accordingly. However, the mechanism to cope with sampled data is not yet implemented.

Unsampled NetFlow information is directly used by BotFinder running on the BotFinder analysis server, the central unit reading in the inputs and raising alarms to the administrator. As will be explained in detail, BotFinder easily parallelizes and scales linearly in the number of servers supporting the analysis by providing computational capacities.

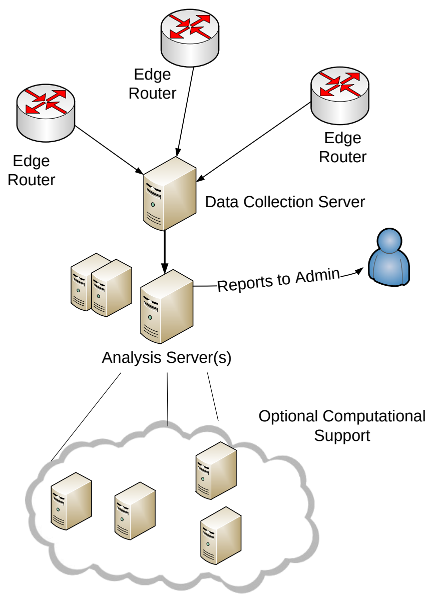


Figure 5.1: BotFinder deployment scenario in a large scale network.

5.2 Implementation Details

BotFinder is implemented as a prototype using the interpreted high level language Python [76]. The abstract processing steps of BotFinder are roughly matched by various scripts as depicted in Figure 5.2. The core script reads input data, assembles traces and, if the length is above the given threshold $|\mathcal{T}|_{min}$, sends the traces to the feature extraction (computation) client. The resulting trace features are written to a file and either piped into the cluster-generation or the model-comparison script. The process is detailed in the following subsections.

5.2.1 Input Data

As BotFinder is able to process packet capture (pcap) or NetFlow data, the traffic collection tool can vary respectively. For full traffic collection classical

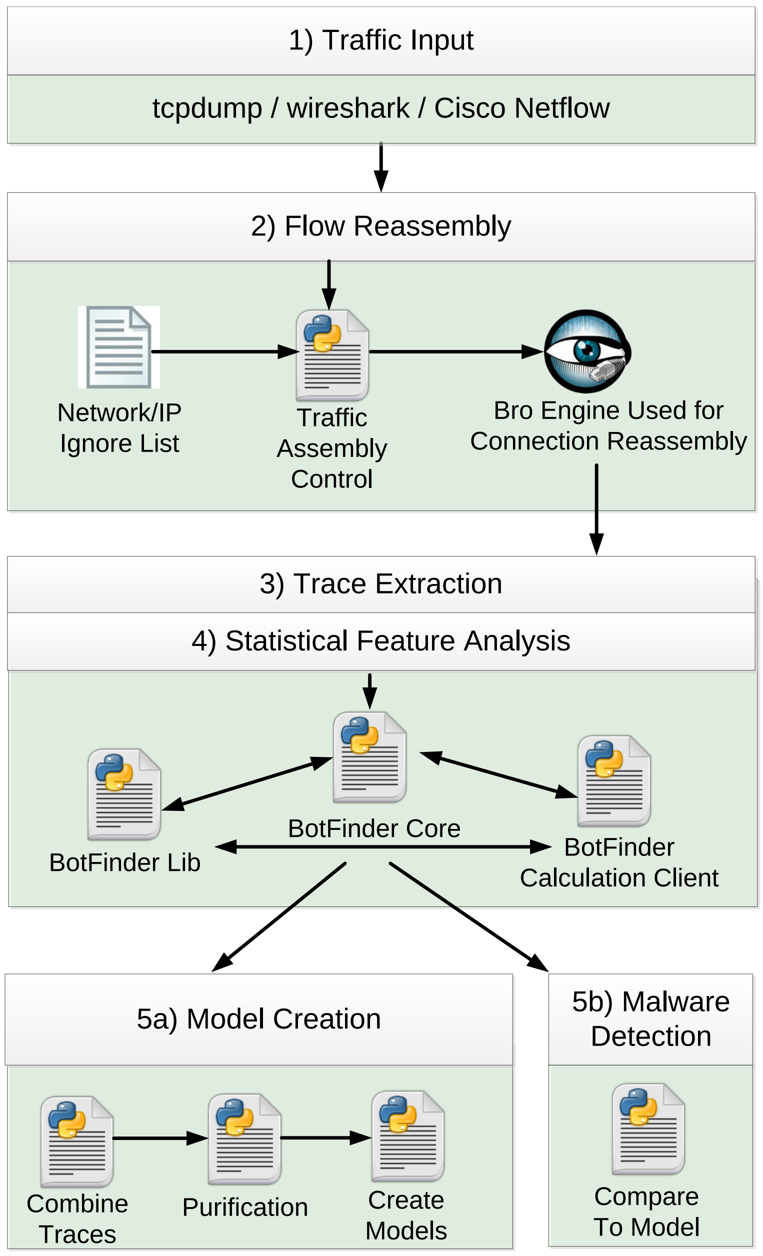


Figure 5.2: BotFinder implementation.

tcpdump[1] or Wireshark[2] can be used (or any other network traffic capturing tool that writes standard pcap files). If Cisco's NetFlow is used, it has to be processed and written to files via flow-tools[3] first, before it is read into BotFinder using pyflowtools[4]. As flow-tools does not support the modular NetFlow version 9, BotFinder works best on the relatively static and widely deployed NetFlow version 5.

5.2.2 Flow Reassembly

In this step, flows are reconstructed from individual packet headers and a whitelisting is applied. In general, BotFinder supports two input formats:

1. For *full packet captures*, the powerful scripting engine of the Bro[5] Network Intrusion Detection System (NIDS) is used for flow reassembly.

2. For *NetFlow* input, no packet level reassembly is required. NetFlow already provides the full flow information collected and processed by the routers.

In the traffic assembly script, a first whitelisting step is applied for which a list of IP addresses or network addresses that should be excluded from further processing is imported. For example, a network administrator may exclude mail traffic (port 25) or whitelisted services. Furthermore, the whole internal IP address-range of the administered network might be excluded to minimize false positives.

5.2.3 Trace Extraction and Feature Analysis

The BotFinder core implements steps 3 and 4 – the trace assembly and the feature analysis – and expects as input a flow level representation of network communication. Based on the tuple of matching source IP address, destination

[1] `http://www.tcpdump.org/`
[2] `http://www.wireshark.org/`
[3] `http://code.google.com/p/flow-tools/`
[4] `http://code.google.com/p/pyflowtools/`
[5] `http://bro-ids.org/`

IP address, protocol ID and destination port, a dictionary structure – the "trace assembly buffer" – is filled with the flows that build traces.

This trace assembly buffer raises special technical challenges for the analysis of large quantities of data: To reassemble sufficiently long traces (Section 6.2), as many connections as possible should be included into the trace. However, if millions or even billions of flows are under investigation in parallel, maintaining the current state of all traces in the trace assembly buffer in memory is hard. Moreover, a live deployment of BotFinder will have to report analysis results in regular intervals. To solve this challenge, the following solutions are implemented.

Balancing the Trace Assembly Load

To process large amounts of flow records in real-time, BotFinder offers various parallelization options that benefit from the fact, that BotFinder performs no horizontal correlation between independent flows.

Multiple Analysis Servers: On a large scale, the data collection server or a special module at the analysis server may balance the trace analysis load among different servers. Hereby, different source IP address ranges may be used to coordinate the load balancing.

Simple Size Limitation: The simplest way to keep the trace assembly buffer in the targeted range, which is typically the overall memory size, is to limit the amount of flows that is read in between each trace calculation step. This method requires the user to specify the amount of traces that should be read in. Statistically, the smaller the trace assembly buffer is chosen, the less long traces will be generated and the lower is the detection quality of BotFinder.

Furthermore, BotFinder offers two different behaviors after reaching and processing a full trace assembly buffer. In the non-overlapping mode, the trace assembly buffer is deleted and the next processing step starts after re-filling the buffer. As an alternative, BotFinder allows to let the trace assembly buffers of each processing interval overlap (e.g., by 25%) to minimize problems at the end of each processing buffer. Nevertheless, simply reducing the size of the buffer comes with a number of disadvantages, such as the potential loss of traces due to small buffer sizes.

File Based Analysis: As the trace building process is linear and requires no interaction with other traces, the processing is easily split by IP addresses and the flows are stored in different files. As an example, a /24 network traffic may be split up to 254 files and each file is processed individually. Thereby, the amount of data that has to be kept simultaneously in the trace buffer is limited.

Especially for offline analysis, the file based approach provides substantial benefits. Conceptually, the number of sub-splitting files is infinite, although technically a limitation of $\approx 2^{10}$ is observed (Ubuntu 11.04). Still, using, for example, 768 files to distribute flows based on the source IP address allows to compute traces over a significantly longer timeframe than to simply use in-memory storage of all traces.

Processing the Trace Buffer

After all files are read or the trace assembly buffer is full, the processing starts. To speed up the process, BotFinder uses multiple threads to separate the work of reading data from the processing workload. Thereby, after reading the buffer, sufficiently long traces are sent to the feature extraction clients and the reading of a new buffer continues while the old buffer is being processed. This concurrency is especially useful if traffic or NetFlow data is directly piped into BotFinder without saving it to files first.

For the feature extraction, BotFinder's performance is inherently impacted by the computational challenge to perform large amounts of Fast Fourier Transforms and other mathematical tests. To cope with this problem, BotFinder supports optional computational workers as highlighted in the blue cloud in Figure 5.1. These calculation clients expect a trace as input and perform all statistical analysis and report the information on averages, deviations, and FFT back to the analysis server. As the processing of one trace does not require any result from the processing of another trace, the process parallelizes well and an arbitrary number of machines can be used for computational support. In an optimal scenario, the processing completes just before the trace assembly buffer is filled again.

All communication with the networked calculation clients – whereby BotFinder allows to run both entities, the core script and the calculation client on the same host – is done in an additional thread that manages multiple processing clients. Hereby, BotFinder already reads new data while processing the buffer. The processing results are written to a file that serves as input either to the model-creation script or the model-comparison (detection) script.

5.2.4 Calculation Client

The calculation client is a simple script that opens a network port and awaits formatted trace-strings. After receival, the processing of traces is started on all available (or configured) cores. Hereby, simple mathematical operations such as the averaging, standard deviation calculation, and summations are done directly in Python. For complex operations such as the Fast Fourier Transform, the statistical computation environment *R* [60] is used. Using the rpy2[6] Python interface, *R* is easily accessed and Python data containers (especially lists) are shifted to *R* in a fast manner.

Two processing elements govern the computation load during the FFT analysis. The first step is the sampling of the trace elements as a binary signal, which may lead to a large number of data points when very small intervals enforce a small Nyquist frequency (see Section 4.4 for details) while large gaps in the trace require many 0 sampling points. If a small frequency would enforce more than 2^{16} datapoints, the sampling frequency is corrected and minor undersampling is accepted. The limitation that the number of datapoints has to be a factor of two directly leads to the second time consuming step during calculation, the actual FFT calculation. *R* implements the routines by Singleton [66] and is fastest when the length of the data series is highly composite.

After calculating the FFT, the three strongest frequencies, their power, and the overall power sum is extracted and returned via the network socket.

5.2.5 Create Model – Clustering

The script to create models reads a trace feature output file and creates models using the CLUES clustering algorithm. Each feature is clustered individually (as described in Section 4.5) using the standard parameters defined by Chang et al. [11] with an increased $\alpha = 0.2$ (from 0.05). Additionally, the number of iterations is set from 20 to 200. All calculations are performed in *R* and the generated clusters are written to file. Additionally to the cluster center and size, the standard deviation and cluster quality is calculated. For the latter, the β parameter as exponent in the quality rating function has to be supplied, or the default value of $\beta = 2.5$ is used. The final clusters are written to file using an XML based data-structure as shown in Figure 5.3. Each cluster element contains the information of cluster size, total size of all clusters, the center,

[6]`http://rpy.sourceforge.net/rpy2.html`

```xml
<pure_pushdo>
    <avgtime>
        <cluster>17,106,376.502597943,10.0249808739,0.935600716224</cluster>
        <cluster>24,106,405.306230591,18.4277554409,0.892556348828</cluster>
        <cluster>32,106,250.248460408,9.13601506973,0.912771737511</cluster>
        <cluster>17,106,214.224754301,28.899597005,0.713724740726</cluster>
        <cluster>16,106,1273.99011928,137.67907301,0.763247471045</cluster>
    </avgtime>
    <srcbytes>
        <cluster>62,106,208.973513213,152.728669969,0.160874693196</cluster>
        <cluster>44,106,132.067745795,27.8883917682,0.589831546777</cluster>
    </srcbytes>
    <dstbytes>
        <cluster>70,106,163.445178164,5.27597132832,0.922470941658</cluster>
    </dstbytes>
    <avgduration>
        <cluster>36,106,9.91748253516,14.750168725,0.0242770450496</cluster>
        <cluster>17,106,0.639529625624,0.0329234397102,0.87923613487</cluster>
        <cluster>18,106,0.543640351578,0.0171900840755,0.923992975088</cluster>
        <cluster>17,106,0.262405375039,0.0470137452867,0.638961282986</cluster>
        <cluster>18,106,0.022240949149,0.0155630677467,0.17388329796</cluster>
    </avgduration>
    <fftfreq>
        <cluster>54,106,0.00554994321007,0.00531778447144,0.0911341281862</cluster>
        <cluster>52,106,0.00200745042759,0.0007946695012,0.371706684647</cluster>
    </fftfreq>
</pure_pushdo>
```

Figure 5.3: XML based representation of the Pushdo model.

the standard deviation of the cluster, and the quality.

To combine multiple traffic samples of the same family, the respective feature files are combined to one large file in an additional script. Hereby, the bot identification is simply based on filename elements that are either automatically generated during sample generation – if BotFinder scripts are used – or manually by the BotFinder operator, if third party malware traces are supplied for training.

The purification script implements the standard and strict modes of purification and reads a trace feature file as input. The binary sample is identified by a MD5 tag that is associated to each sample trace. A list containing IP addresses and network masks of identified *C&C* servers is used as additional input to the purification script. The overall structure of files after trace extraction, combination and purification remains the same. The steps of combination and purification are optional processing steps.

5.2.6 Compare to Model – Detection

The model comparison script reads a model file and the trace feature file that should be investigated for infections. As described in Section 4.6, each feature is matched individually and for each model a separate γ_m is maintained. For each feature, the script checks whether the trace average is in the range of plus-minus two standard deviations around the cluster center. The best matching model is chosen and, if $\gamma_m > \alpha$, a report line is written. Hereby, additional statistical information such as the contributing features and the γ_m value are presented.

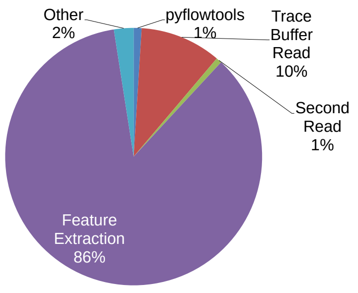
5.3 Performance

Although BotFinder's implementation is on a prototype level, a qualitative and quantitative evaluation sheds light on time-costly computational processes and the real world applicability of BotFinder. As a prototype, BotFinder is implemented in Python, which is – as an interpreted language – significantly slower than a native language like C [40]. Further, the Fast Fourier Transforms are performed via the statistical computing environment *R* [60] and the Python interface rpy2[7]. This shift of data from Python to *R* is hindering the performance as well.

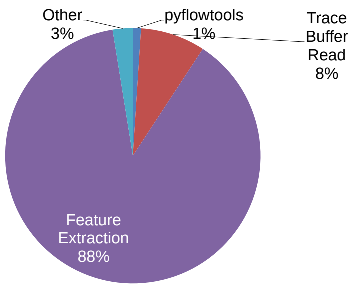
In general, BotFinder's performance is systematically impacted by three factors:

1. The choice of input data which is either a Bro pre-processed file or a flowtools packed NetFlow records.
2. The choice of trace buffer management: If the optimization for large amounts of data is used and the data is read and distributed towards more than 700 files, these files have to be written and read again which impacts the performance. If all data is kept in-memory, which is only possible for few Gigabytes of input data, the input file has to be read only once.
3. The number of CPU cores available for feature extraction: For simplicity reasons, all experiments – if not explicitly stated otherwise – are run using a single core.

[7]`http://rpy.sourceforge.net/rpy2.html`



(a) File based



(b) In-Memory

Figure 5.4: Workload distribution among different BotFinder tasks using one CPU core.

Further, hardware factors like the read and write speed of the harddisk, the CPU speed and the memory size and speed are relevant for the overall performance. All experiments are conducted on a Dell lab machine with an Intel Core i7 CPU with 8 cores at 2.8GHz and 12GB of RAM.

Figure 5.4 illustrates the workload distribution of BotFinder to read 15.3 million lines of randomly chosen NetFlow data using files (Figure 5.4(a)) and the in-memory trace buffer (Figure 5.4(b)). As can be clearly seen, the overall processing is dominated by the feature extraction step (86% and 88%). Using one CPU, the processing of the 15 million lines of NetFlow requires $\approx$ 1690 seconds using the file based approach and $\approx$ 1640 seconds using the in-memory method. The pyflowtools package performs relatively well using 16.83 seconds in total or 0.16 seconds per 100,000 lines of NetFlow. Please note that the NetFlow data can be expected to produce a relatively low amount of long traces and therefore relatively low feature extraction overhead, as millions of hosts contribute to the dataset. For other datasets, e.g., the *LabCapture* dataset, the feature extraction workload is higher per line of input flows.

Although the file read, write and re-read time is increased by 37% for the file based approach, the difference is of low overall relevance considering the domination of the feature extraction time. The general processing accounts for only 2.4% of the overall run time.

If four CPUs are used, the processing time is reduced significantly to 876 seconds which corresponds to nearly perfect linear scaling in the feature extraction domination. Here, the complete independence of each trace during statistical analysis is advantageous to achieve good parallelization.

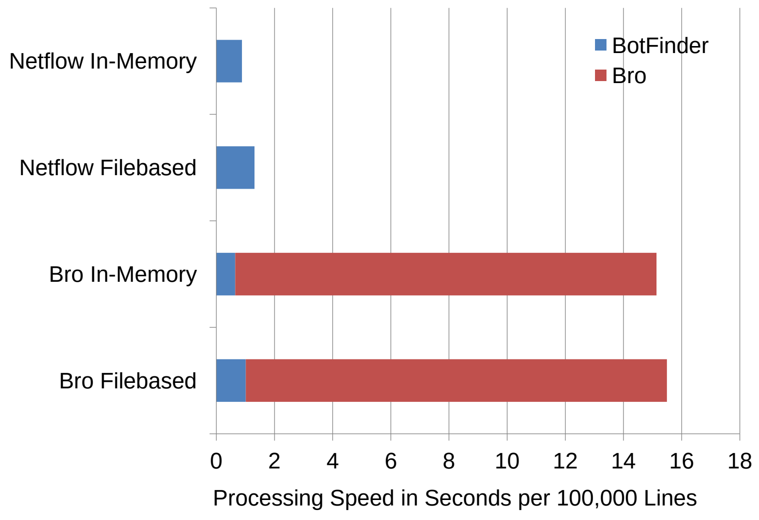


Figure 5.5: Input data processing using file based and in-memory solutions. For Bro input data, the pre-processing using Bro has to be added to BotFinder's processing.

Using as many calculation clients and cores as possible reduces the feature extraction speed significantly. However, as the creation of the trace assembly buffer hardly parallelizes, the input processing becomes the major bottleneck. Figure 5.5 depicts the absolute processing speed to read 100,000 lines of NetFlow or Bro pre-processed data.

BotFinder reads 100,000 lines of NetFlow in 0.88 seconds in in-memory mode and in 1.31 seconds using files to handle larger inputs. If data is used that was pre-processed using Bro, BotFinder reads 100,000 lines of Bro output in 0.65 seconds (in-memory) or in 1.01 seconds (file based). However, the pre-processing has to be added to the overall processing time. Figure 5.5 compares the different processing times and Bro impacts the performance substantially. In a measurement experiment, Bro extracted 621,362 flows from 3GB of raw network pcap data in $\approx$ 90 seconds. As a rough number, one can state that Bro takes around 14 seconds to generate 100,000 flows. Still, the Bro-process is influenced by the size of the flows. If few flows that transferred a large amount of Bytes are injected to Bro, the per flow processing time increases.

In summary, BotFinder performs fastest if the high computational load of feature extraction is outsourced to a larger amount of processing clients, which makes the input processing the most relevant key for time saving. Hereby, data that was pre-processed by Bro is computed fastest, but the Bro computation time has to be considered. As the difference between in-memory processing and the file based approach is low, the recommendation for real world deployment of BotFinder is to use the file based approach and NetFlow data as input.

Chapter 6

Parameter Analysis

This chapter analyzes the influence of the individual BotFinder parameters. A challenging factor is the high inter-dependency between the different parameters. For example, a high correlation between the minimal trace length and the applied purification method is observed. Each parameter is investigated in an isolated way by varying a single parameter and fixing the remaining. Parameters and methods under investigation are:

- The purification method, especially the amount of trace reduction and the final impact on clustering quality and detection rates.
- The minimal trace length for trace consideration. This factor is intended to reduce the number of false positives by statistically weak, short traces and, implicitly, reduce the workload for BotFinder.
- The error-aware detection method judging the quality of each individual trace that is under investigation. Hereby, the δ parameter controls the individual trace quality and contribution.

6.1 Minimal Cross-Validation Experiment

As a basis for further parameter analysis and optimization, the first experiment conducted is a cross validation run with a minimal set of parameters. This cross-validation experiment can already be counted towards the evaluation of BotFinder: In this experiment, the BotFinder prototype is applied to (a) traffic produced by the malware samples described in Section 2.4 and (b) the *LabCapture* traffic. For each varying acceptance threshold α, 30 independent cross-validation runs were executed as follows:

1. This thesis' ground truth malware dataset, as given in Table 6.1, is randomly split into a training set $\mathcal{W}$, which contains 70% of the traces, and a detection set $\mathcal{D}$, which contains the remaining 30% of traces.

2. The set $\mathcal{D}$ is mixed with the traces from the *LabCapture* dataset, which is assumed to be completely infection-free, and is therefore a reasonable dataset to derive the false positive ratio of BotFinder.

3. Thereafter, BotFinder is trained on the bot behavior exhibited in the traces in $\mathcal{W}$ and creates the models accordingly.

4. Finally, the generated models are applied to the mixed set composed from $\mathcal{D}$ and the *LabCapture* dataset.

The analysis is performed on a per-sample level, which is possible as the information which malware binary generated a specific trace is available. More precisely, if one trace of a sample is correctly identified by a trace match, the entire sample is counted as correctly identified; if a trace of a given malware is classified as a different malware, this match is considered as a false positive.

No purification is applied and all traces from the malware samples are used for detection. The minimal trace length is set to 5, ensuring, that even short traces are used for analysis. Still, the 75% short connections (see also Section 3.5.1), flows to port 25, and whitelisted connections to, for example, the Windows Update Server, are removed. No input trace error awareness (the δ parameter) during detection is used. The cluster quality rating function requires a parameter β, which is initially set to 2.5. The requirement to hit at least h features to raise an alarm is disabled.

Figure 6.1 shows the cross-validation detection rate and the corresponding false positive ratio. Please note, that the false positives are, as throughout the whole thesis, depicted in logarithmic scaling. For a detection ratio of around 75%, a false positive rate of 0.0015 is obtained. In other words, for 10,000 traces, 15 false positives are raised. Considering, that all traces longer than 5 flows are under investigation, a typical network with millions of traces raises a large amount of false positives.

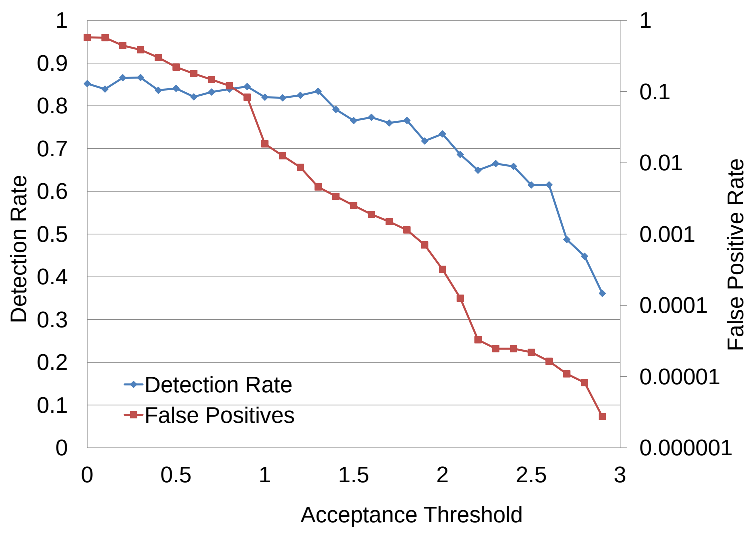


Figure 6.1: Detection rate with minimal parameter setting.

Starting from this baseline of detection and false positive rates, the introduced parameters are investigated for their impact on reducing the number of false positives while maintaining a high detection ratio.

6.2 Data Purification and the Minimal Trace Length

The data purification method is, as aforementioned, an optional step to optimize the input data for BotFinder's training. The purification is related to the minimal trace length, as *C&C* connections have a relatively high connection frequency to the *C&C* server. This high frequency is required for the bots to remain agile members of the botnet.

To highlight the correlation between trace lengths and the purification sets, the trace lengths distribution for three different purification levels and the trace length CDF is shown in Figure 6.2. The three levels are as introduced in Section 3.5.2:

1. *No purification* is applied, therefore all traces are in the training set.

2. The *standard* approach: Effectively, if a sample has matching traces, all non-matching traces of that sample are dropped. If, on the other hand, a sample has no matching traces, the entire set of traces is used. For this *C&C* IP address matching, an existing, UCSB in-house-managed *C&C* control server list is used. Using various other lists, such as EmergingThreats[1], which is based on information from Shadowserver[2], Spam-Haus[3], and DShield[4], yields similar results.

3. The *strict* approach, whereby only confirmed (black-list matching) *C&C* traces are used. Samples that do not exhibit a connection to a blacklisted IP address are ignored.

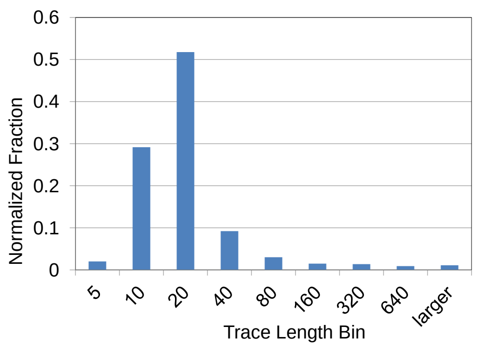
6.2.1 Matching IP Addresses

The standard purification step yields a reduction of training traces as shown in Table 6.1. On average the amount of traces is reduced to 40%. For Banbra, 24% of all samples had a blacklist-matching trace but the overall amount of sufficiently long traces is not reduced as the standard approach adds all traces of a non-matching sample to the training set. For Bifrose, the purification impact is minimal as well, as only two traces are removed, but 85% of the samples exhibit traffic to blacklisted IPs. This supports the assumption that the long traces are actually the malicious *C&C* communications. For Blackenergy, 34 samples exhibited 74 connections, whereby only 12% of the targets were known, which explains the low reduction of traces by 7. Sasfis is similar to Banbra as only 36% of the samples connected to a known *C&C* server but all traces are used after standard purification. Each sample only exhibited one trace of sufficient length.

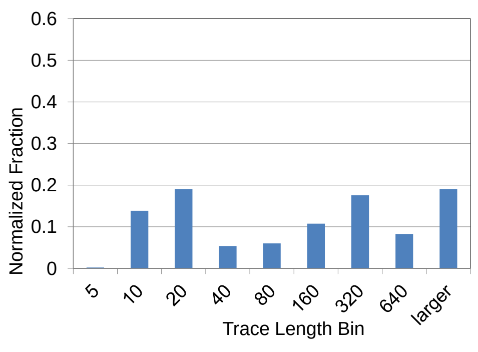
Especially Dedler benefitted from the purification step as 395 traces are reduced to 46. This corresponds to two different *C&C* server traces per sample. Each Dedler sample exhibited connections to known malicious servers.

A significant reduction of traces is also achieved for Pushdo, where 190 traces of 55 samples are reduced to 106 traces. Overall, 64 percent of all samples connected to a known *C&C* server.

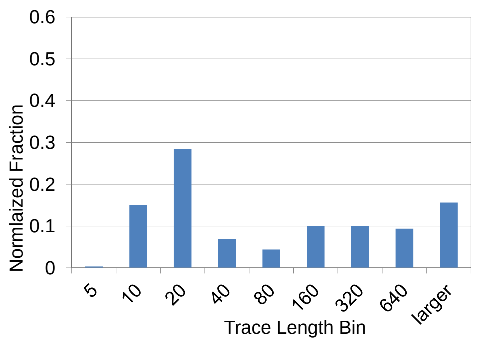
[1]`http://rules.emergingthreats.net/`
[2]`http://www.shadowserver.org`
[3]`http://www.spamhaus.org`
[4]`http://www.dshield.org`



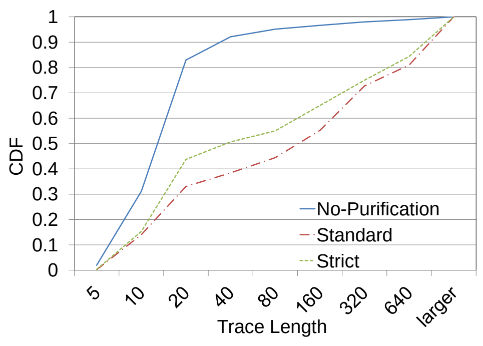
(a) No purification.



(b) Standard approach.



(c) Strict approach (*C&C* only).



(d) Trace length CDF.

Figure 6.2: Trace length distribution.

Family	Samples	Total Traces	Traces after Purification	C&C hits in Blacklist	CQ w P.	CQ w/o P
Banbra	29	29	29	0.24	0.99	0.99
Bifrose	33	31	31	0.85	0.52	0.52
Blackenergy	34	74	67	0.12	0.47	0.57
Dedler	23	395	46	1.00	0.39	0.76
Pushdo	55	190	106	0.64	0.55	0.49
Sasfis	14	14	14	0.36	0.88	0.88
Average	32	122	49	0.54	0.63	0.70

Table 6.1: Malware families used for training. Purification is especially successful if many traffic samples show traces matching blacklisted IPs. A high quality indicates a low standard deviation within the clusters.

6.2.2 Trace Length and Purification Correlation

To illustrate the high correlation between trace length and the property to be a *C&C* trace, lengths distributions for the different purification levels are shown in Figure 6.2.

The first set consists of non-purified traces and is therefore build from all traces of length 5 or larger – in total 11,046 traces. The histogram in Figure 6.2(a) uses nine bins of exponentially increasing size and shows a peak in the trace length region below 40 flows per trace. The CDF in Figure 6.2(d) highlights this by showing that 83% of all traces are of length ≤ 20, and 92% have ≤ 40 flows per trace.

Applying the standard approach of purification significantly impacts the distribution of trace lengths and their overall quantity, which is down to 490 traces or 4.4%. As Figure 6.2(b) shows, longer traces are contributing significantly more to the overall trace set. Only 38% of all traces are shorter than 40 flows and the majority of traces is longer than 50. For the strict purification level, the distribution remains tending towards higher trace lengths (Figure 6.2(c)), but the CDF reveals that "only" 50% of the remaining 326 traces are of length 40 or longer. The explanation to this slight shift to shorter traces is the incompleteness of the used blacklists, which is unavoidable in real life environments. Especially the Banbra malware samples exhibited highly regular and long traces but connected to two different *C&C* servers. The IP address of only *one* server was known so that roughly 70% of these traces are missing in the strictly purified dataset.

As a first result it can be stated that the minimal trace length $|\mathcal{T}|_{min}$ is highly related to the trace's property to be a *C&C* communication trace. For the training data of the six malware samples used in this thesis, the probability to be a *C&C* trace if above length 50 is about 20%. Ignoring the vast amount of Dedler traces and considering the remaining five malware families, the probability is more than 65%.

6.2.3 Detection Rate and False Positives

For the detection quality, the impact of purification and minimal trace length variation is depicted in Figure 6.3. The first three figures (6.3(a)-6.3(c)) depict the detection rate based on the purification level and the three minimal length parameters. Figures 6.3(d)-6.3(f) illustrate the false positive rates in

	$\|\mathcal{T}\|_{min}$	No Purification	Standard	Strict
Count	10	8499	2044	733
Quality	10	0.579	0.628	0.648
Count	20	438	326	293
Quality	20	0.650	0.685	0.770
Count	50	289	180	154
Quality	50	0.788	0.822	0.869

Table 6.2: Number of traces available for clustering and the clustering quality.

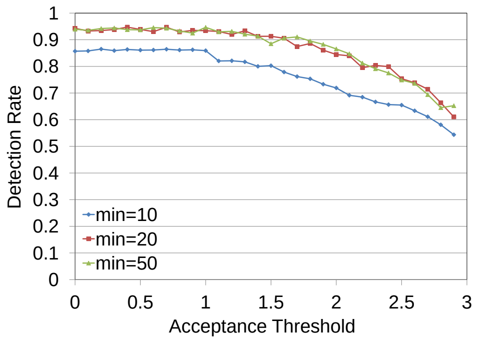
logarithmic scaling.

Generally, better detection rates correlate with higher minimal trace lengths. This is consistent with the aforementioned high probability for a long trace to be a *C&C* trace. Therefore, even without purification (as depicted in Figure 6.3(a)) high detection rates are achieved when the minimal trace length is set to $|\mathcal{T}|_{min} = 50$. Similar detection rates are obtained for the standard purification system but lower rates for the strict approach. This is counter-intuitive as the strict approach only consists of confirmed *C&C* traces. However, by dropping many actual *C&C* traces that connect to not-known *C&C* servers, less traces contribute to malware clusters. If these missing traces actually were of high quality as, e.g., the Banbra traces that connected to a not-known *C&C* server, good traces are not counted in the strict purification level. Therefore, a stricter purification level can in fact decrease the overall performance.

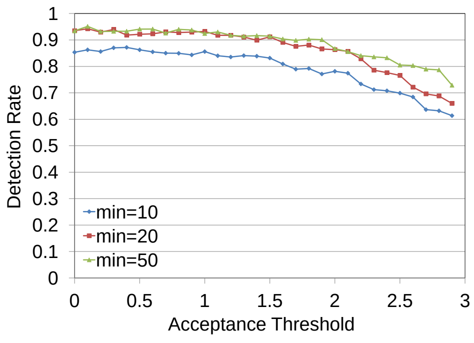
Furthermore, as Table 6.2 illustrates, the cluster quality increases with increasing minimal trace length requirements and increasing purification strictness. However, the total number of input traces reduces. Interestingly, the false positive levels are only marginally impacted by such different cluster qualities. This indicates that the exponentially decreasing impact of weaker clusters reduces the effect of loose clusters.

6.2.4 Purification Independence and Cluster Quality

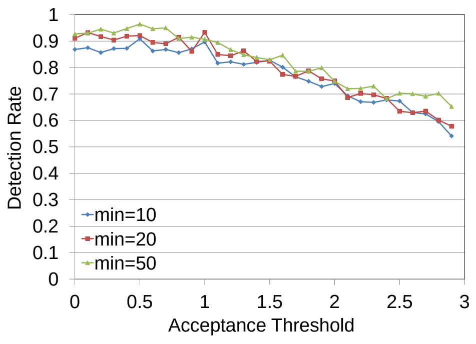
It is important to note that BotFinder is neither depended on the purification step nor does a low purification ratio indicate weaker models. Even when not many connections to known *C&C* servers are found, this does not necessarily result in lower quality models: For example, all 29 samples of Banbra connect to only two different destination IP addresses and only one address



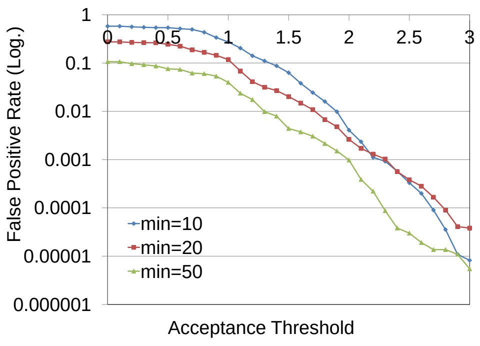
(a) No Purification.



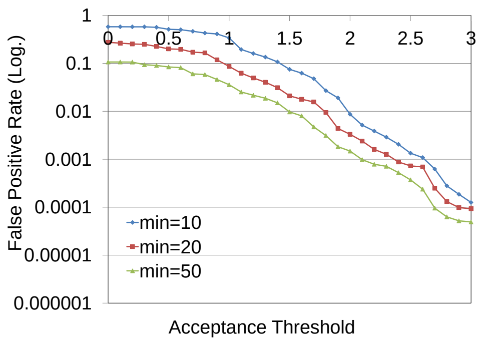
(b) Standard Purification.



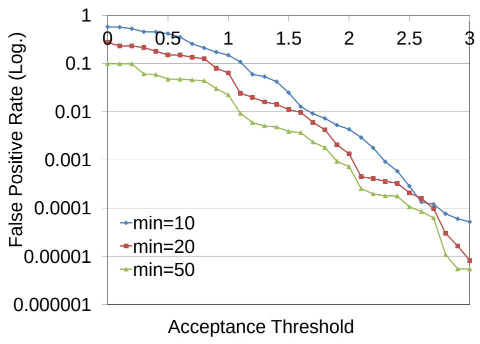
(c) Strict Purification.



(d) No Purification.

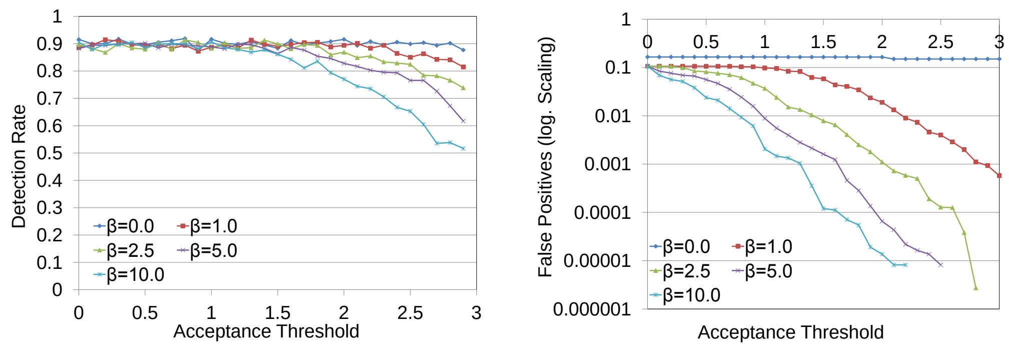


(e) Standard Purification.



(f) Strict Purification.

Figure 6.3: Detection rates and false positives based on purification method and minimal trace length.



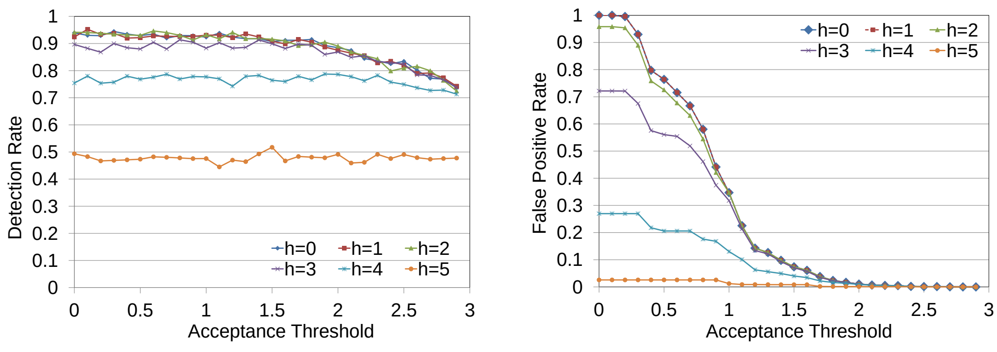
(a) Detection Rate (b) False Positives

Figure 6.4: Influence of the cluster quality control parameter β.

was known to the *C&C* blacklist used for purification. Still, the traffic pattern and periodicity features are highly similar which leads to a very high Clustering Quality (CQ) based on the exponentially decreasing quality rating function of 0.99. This highlights that BotFinder does not require a data purification process. In particular, as can be seen in Table 6.1, the difference between the cluster qualities with and without purification is small (or zero) for most families. A main exception is Dedler, for which the purification results in a significantly better cluster quality which improved from 0.39 to 0.76. The large number of potentially benign traces that are dropped for that specific malware reduce noise and optimize the clustering.

6.3 The Cluster Quality Parameter

The β parameter allows to control the exponential decrease of the cluster quality rating function. Its purpose is the reduction of false positives balanced with a high detection rate. Figure 6.4 shows the impact of varying cluster quality parameters $\beta = 0, \beta = 1.0, \beta = 2.5, \beta = 5.0$, and $\beta = 10.0$ for standard purified data with respect to the detection acceptance threshold α. The detection rate remains stable until $\alpha \geq 1.5$ and decreases slowly for further increasing α. The larger β is chosen, the larger is the decrease of detection. Using $\beta \leq 2.5$ allows detection rates greater than 85 percent. However, the impact of β on the false positives is significant in the order of magnitudes. For $\beta = 0.0$, which reflects an increase of the model-matching scoring variable γ (see Section 4.6.2) by 1.0 per cluster hit, only a minimal step at $\alpha = 2.0$ is observed. However, a false



(a) Detection Rate (b) False Positives

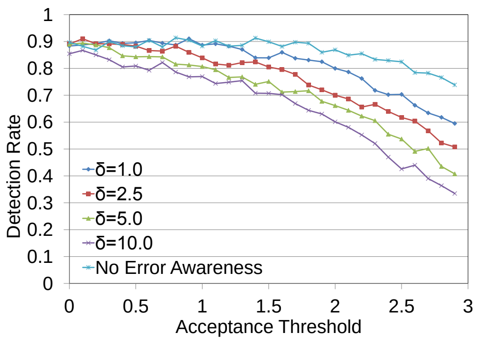
Figure 6.5: Influence of the minimal feature hit requirement h.

positive rate for $\beta = 0.0$ of 16 percent is far from real world usability. In the region of $\alpha = 1.5$, increasing β roughly reduces the false positive rate by an order of magnitude.

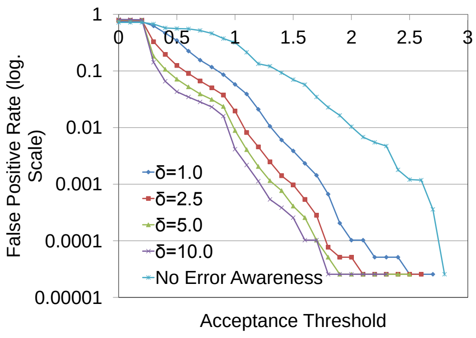
The selection of an appropriate β is hard as one has to balance the decrease of the detection rate with the intended reduction of false positives. For further experiments, a threshold of $\beta = 2.5$ is selected that only minimally affects the detection rate but reduces the false positives to 1% compared to a calculation with $\beta = 0.0$.

6.4 The Minimal-Hits-Requirement

The introduction of an additional requirement for BotFinder to raise an alarm is intended to reduce false positives that occur by having a benign trace accidentally match one or two clusters. For standard purified training traces, the detection rates and false positive rates for $h = 0, .., 5$ are analyzed using $\beta = 2.5$ and a minimal trace length of 50. As Figure 6.5(a) illustrates, the detection rate is relatively high for $h \in [0, 3]$ and degrades for further increasing h. The effect on the false positives is shown in Figure 6.5(b) in linear scaling. h impacts the false positive rate only minimally for low values $h \leq 3$, but becomes significant for larger values. However, the effect is highly correlated to the α threshold and especially impacts results with lower α, as higher α are only reached by traces that hit multiple clusters anyways. As a default for BotFinder, $h = 3$ is chosen, which yields similar detection results than low



(a) Detection Rate



(b) False Positives

Figure 6.6: Influence of the δ parameter that rates the quality of traces in the detection phase.

h but reduces the false positive rate for low α.

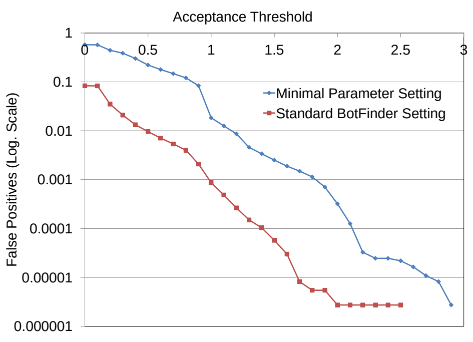
6.5 Trace Input Error Awareness

The introduction of an error factor δ for the traces under investigation is intended to decrease the amount of false positives (as *C&C* traces are assumed to be quite periodic) and to sharpen the detection results.

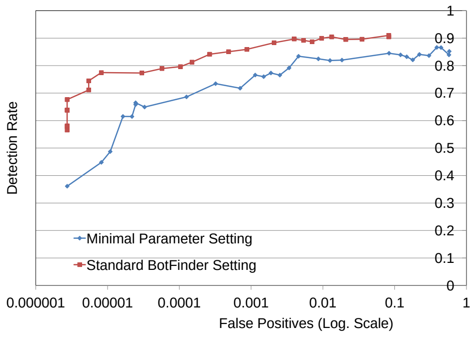
Figure 6.6 shows the detection and false positive rates for four different δ values of $1.0, 2.5, 5.0$ and 10.0. The minimal hit requirement parameter is kept invariant at $h = 3$ and the standard purification method is used.

As can be seen in Figure 6.6(a), the detection rate is impacted by increasing δ, but remains relatively high. For $\delta = 2.5$ the detection rate remains above 70% for acceptance thresholds up to $\alpha = 2.0$. However, as intended, the false positive ratio significantly decreases compared to the false positive rate without error consideration. Especially for higher acceptance thresholds in regions of $\alpha \geq 1.5$, the false positive rate is reduced to $\approx 0.8\%$ on average.

In summary, the consideration of the quality of the trace under investigation yields similar detection rates, but significantly reduces false positive rates. As a balance between detection rate reduction and false positive impact, $\delta = 2.5$ is chosen as the default value for BotFinder.



(a) False Positives.



(b) Detection Rate vs False Positives.

Figure 6.7: Impact of the BotFinder parameters. All false positive values are plotted on a logarithmic scale.

6.6 Overall Impact

Each additional parameter supports BotFinder's detection rate and effectively lowers the false positives. Combining all parameters to the general BotFinder setting of $\beta = 2.5$, $\delta = 2.5$, $h = 3$, and a minimal trace length of 50, significantly lowers the false positives ratio while the detection ratio remains stable. Still, all introduced parameters utilize the higher periodicity of real world *C&C* traces.

Figure 6.7 illustrates the total effect of the parameters chosen. The plots show that the parameters introduced efficiently reduce the false positives rate: Figure 6.7(a) depicts the false positive ratio with varying acceptance threshold α. Although both curves show the same behavior of exponential degradation – near linear in the logarithmic plotting – the overall distance is significant: At the typical acceptance threshold range of BotFinder ($\alpha \in [1.5, 2.0]$), the false positive ratio using all parameters is of only 0.4% of the one using no optimizations. This is an improvement of *three orders of magnitude*. Figure 6.7(b) shows the detection rate plotted against the false positive rate in logarithmic scaling. A value near to the upper left corner is optimal (this is for $\alpha \in [1.5, 2.0]$) as it combines high detection rates with low false positive ratio. At all times, the standard settings of BotFinder outperform the minimal parameter settings. The curve is shifted towards better detection rates while generating less false positives.

Chapter 7

Evaluation

To evaluate BotFinder, a number of experiments on the two datasets, *LabCapture* and *ISPNetFlow*, is performed. As presented in Section 2.6, the *LabCapture* dataset is a full packet capture of 2.5 months of traffic of the UCSB security lab with approximately 80 lab machines and it should – by lab policy – be malware trace free. As the full traffic capture is available, a manual verification of bot reports can be executed. The *ISPNetFlow* dataset covers 37 days of NetFlow data collected from a large network. Although no ground truth for the second network dataset is available, as the underlying, full traffic capture required for full content inspection is not available, relevant information can be obtained from the experiments: The identified hits can be compared to known malware IP blacklists and the usability of the approach for the daily operation of large networks can be judged.

Using the prototype implementation presented in Chapter 5, the following experiments (for a detailed description please refer to the respective subsections) were performed:

1. A cross-validation experiment based on the ground truth training data and the *LabCapture* dataset as already used in Chapter 6: In short, the training data is split into a training set and a detection set. The latter is then mixed with all traces from the *LabCapture* data that should not contain bot traces. After BotFinder has learned the bots' behavior on the training set, the detection ratio and false positives in the dataset that contained both the remaining known malicious traces and the *LabCapture* data is analyzed. [**Section 7.2**]

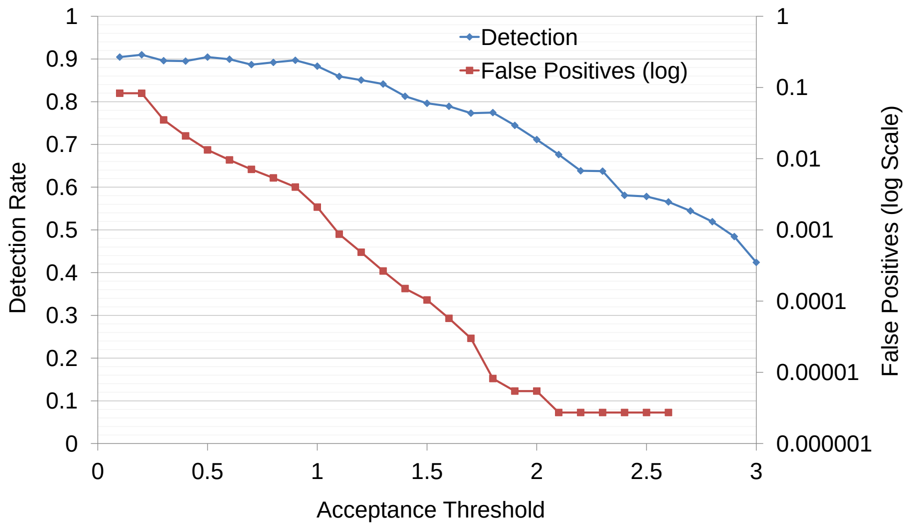


Figure 7.1: Detection Rate of BotFinder in cross validation experiments.

2. Comparison to related work: Here, the most relevant related work is the well-known, packet-inspection-based system *Bothunter* [31]. All experiments are performed on a set of a fraction of ground truth *C&C* traces mixed with the *LabCapture* dataset. [**Section 7.3**]

3. *ISPNetFlow* analysis: BotFinder is trained on all malware training traces and applied on the *ISPNetFlow* dataset in daily slices. The identified malicious traces are investigated and compared to blacklisted malicious *C&C* server IPs. [**Section 7.4**]

7.1 Training BotFinder

BotFinder is trained on the six representative malware families of Banbra, Bifrose, Blackenergy, Dedler, Pushdo and Sasfis as presented throughout this thesis. A standard purification step (for details see Section 3.5.2) was performed and results to, on average, 76 traces per malware sample that are used for model creation. The model generation is based on the features extracted from the selected traces.

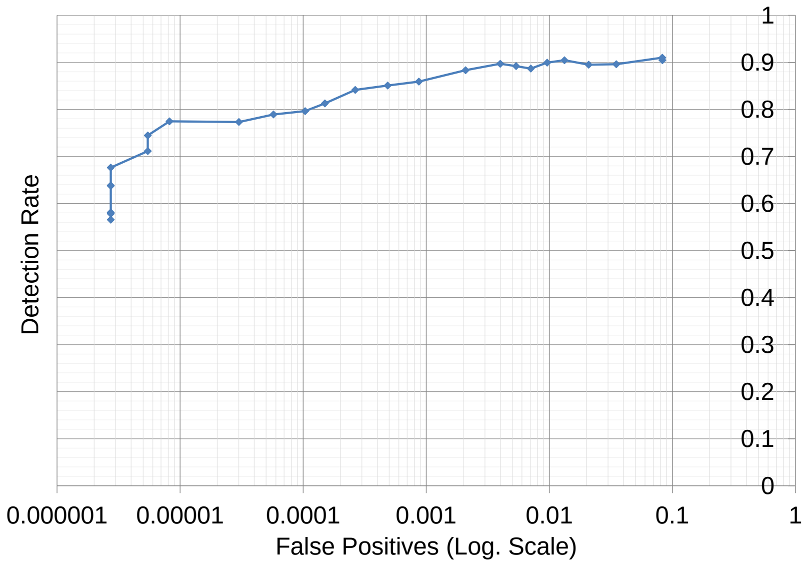


Figure 7.2: Detection Rate of BotFinder in cross validation experiments plotted against logarithmic false positives rate. Optimal values approach are found in the higher left region.

To perform the clustering step, the CLUES algorithm with default values as described in Chang et al. [11] with the aforementioned modifications – to increase the maximum of allowed iterations from 20 to 200 and α from 0.05 to 0.2 – was used. The clustering completes after less than ten seconds and generates on average 3.14 (median 3.00) clusters per feature for each family. Most clusters show a relatively low standard deviation within the cluster, which indicates – again – that the core assumption of BotFinder holds: Different binaries of the same malware family produce similar *C&C* traffic, and this traffic can be effectively described using clustering techniques.

7.2 Cross Validation

The cross validation experiment is executed exactly as described in Chapter 6. A fraction of the training traces (30%) is mixed with the *LabCapture* traces and used for detection. The models are created on the remaining (70%) of malware traces. Following the results from Chapter 6, the parameters are set to $\beta = 2.5$, $h = 3$, $|\mathcal{T}|_{min} = 50$, $\delta = 2.5$, and the acceptance threshold α is varied throughout the experiment.

For each $\alpha \in [0, 3]$, $n = 50$ cross validation runs are executed yielding the results shown in Figures 7.1. Please note that the detection rate is depicted in linear scaling whereas the false positive (secondary) axis is logarithmic. Figure 7.2 shows the detection rate on the y-axis versus the false positive rate in logarithmic scaling on the x-axis.

As can be seen in Figure 7.1, very low acceptance thresholds yield high detection rates of above 90%, but with high false positives. For example, the false positive rate was greater than 1% for $a \leq 0.6$. But, the false positive rate decreases exponentially (near linear in logarithmic scaling) whereas the detection rate decreases roughly linearly. This yields to a good threshold of $a \in [1.7, 2.0]$ – compare the upper left corner of Figure 7.2 – with good detection rates and reasonably low false positives.

For an acceptance threshold of $a = 1.8$, 77% detection rate with $5 \cdot 10^{-6}$ false positives is achieved. For this parameter, Table 7.1 shows the detection rates of the individual malware families, averaged over the 50 cross-validation runs. Here, all Banbra samples and $\approx$ 85% of the Blackenergy, Pushdo and Sasfis samples were detected. The only false positives were raised by Blackenergy (2) and Sasfis (1).

Another interesting experiment is the analysis of the *LabCapture* dataset in daily intervals, similar to a system administrator checking the network daily. More precisely, the traffic capture is split into separate one day slices and analyzed with BotFinder. Overall, 14 false positives – 12 Blackenergy and 2 Pushdo – are observed over the whole 2.5 months time span.

7.3 Comparison to *Bothunter*

In this experiment, BotFinder is compared to its closest related work, *Bothunter* [31]. Hereby, a significant difference is introduced by BotFinder being a content-agnostic malware detection system, whereas the well-known *Bothunter* is a sophisticated bot detection system that relies on a substantially-modified Snort[1] intrusion detection system for flow (dialog) identification combined with anomaly detection. The authors of *Bothunter* later released *Bot-*

[1]`http://www.snort.org`

Malware Family	BotFinder Detection	BotFinder False Positives	*Bothunter* Detection
Banbra	100%	0	24%
Bifrose	49%	0	0%
Blackenergy	85%	2	21%
Dedler	63%	0	n/a
Pushdo	81%	0	11%
Sasfis	87%	1	0%

Table 7.1: Detection rate and false positive results of BotFinder (acceptance threshold a=1.8) in the cross-validation experiment and compared to *Bothunter*.

Miner [30], which adds horizontal correlation between multiple hosts (which is not in scope of this thesis).

The decision to compare BotFinder to *Bothunter* is based on its applicability and the fact, that – to the best of the authors knowledge – no other system allows content agnostic malware detection of individual infections. An exception, under certain conditions, is BotTrack [24]. However, works like BotTrack have a different focus than BotFinder and they detect Peer-To-Peer (P2P) based botnets similar to BotGrep [52] by observing a bot's traffic that is caught in a honeypot. By P2P-Distributed Hash Table (DHT) enumeration, other botnet members are derived by network analysis.

BotFinder is compared to the production version 1.6.0 of *Bothunter*, which is made publicly available[2] by the authors. *Bothunter* leverages detection mechanisms on the whole infection and malware execution life-cycle: Port scanning activities and dangerous binary transfers (e.g., encoded or encrypted HTTP POSTs or shell code) are used to detect the first step of the infection process. Malware downloads ("egg downloads") and, eventually, structural information regarding the *C&C* server plus IP blacklisting of multiple list providers are used to identify infected hosts.

For the comparison, *Bothunter* is run on full traffic dumps of the training samples and the *LabCapture* dataset. The system was installed strictly following the User Guide[3] and configured for batch processing. This *Bothunter* version was chosen as its release time (summer 2011) fits the time of execution of the

[2]`http://www.bothunter.org/`

[3]`http://www.bothunter.net/OnlinePDF.html`

LabCapture traffic capture and approximates the execution time of our malware samples. As can be seen in Table 7.1, very few alarms were raised by *Bothunter* for the training samples. The detection rate varies between 0 and 24 percent for the different families, and a high dependency on IP blacklisting for successful detection is observed. Note that *Bothunter* had access to the full payload for the experiments, whereas BotFinder only operates on flows.

Regarding the test with *Bothunter* on the mixed dataset, alarms were received for 41 distinct IP addresses. For four IP addresses with a significant peak of alarms, a manual confirmation of a bot infection was possible, as researchers executed malware in a bridged VM. As BotFinder was not trained for the specific bot family that the researcher was working on, it is not surprising that BotFinder missed this infection. For most IP addresses (37), a manual confirmation for an actual infection failed. However, manual malware confirmation is challenging and a failed confirmation does not exclude the possibility that actual bot traffic was observed. Nevertheless, often the connections were made to Internet Relay Chat (IRC) servers (BitCoin trades) or alarms were raised because of the high NXDOMAIN activity of the University's core router. In another instance, *Bothunter* identified the download of an Ubuntu Natty ISO as an exploit (Windows Packed Executable and egg download). This shows that the number of false positives is significantly higher than those raised by BotFinder on the same traffic.

7.4 ISPNetFlow Analysis

The *ISPNetFlow* dataset is, as aforementioned, most challenging to analyze as not much information about the associated network and no full packet capture is available. For this experiment, BotFinder is trained on all available training malware traces and applied on the *ISPNetFlow* dataset.

Overall, BotFinder labeled 542 traces as evidence of bot infections, which corresponds to an average of 14.6 alerts per day. This number of events can be easily handled by a system administrator, manually during daily operations or by triggering an automated user notification about potential problems. Figure 7.3 shows the evolution of infections over the analysis time frame, which varies from days with no infections at all to days with a maximum of 40 reported infections. Figure 7.4 illustrates the total number of reported incidents per bot. Pushdo and Dedler are dominating the detected infections with 268 and 214 reports, respectively, followed by Sasfis with 14 and Blackenergy with 12.

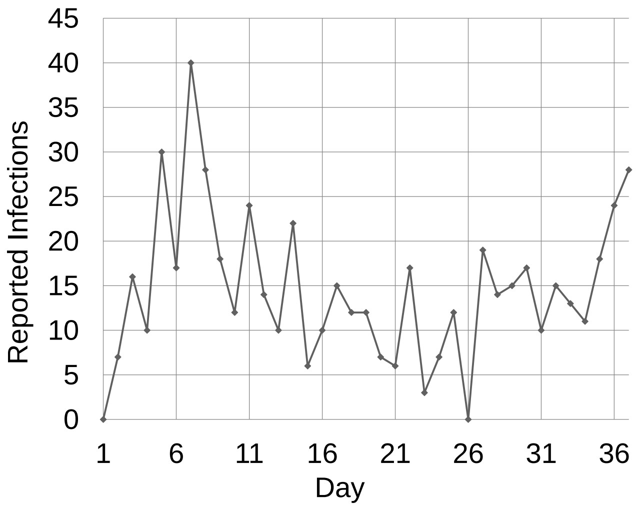


Figure 7.3: Reported infections over time.

Bifrose was found only once in the traffic and Banbra was not found.

To judge the detection quality of BotFinder in the *ISPNetFlow* dataset, the IP addresses involved in suspicious connections were investigated in detail. After receiving the internal IP ranges from the *ISPNetFlow* system administrators, the set of involved IPs was split into internal and external IP addresses.

Only two out of the 542 traces had their source and destination IP addresses both inside the network. This indicates that BotFinder is not generating many – most probably false – indications of internal infections where both the bot and the *C&C* server are inside the observed network.

The remaining 540 external IP addresses are compared to a number of publicly available blacklists[4] and *had a positive match for 302 IPs or 56%.* This result strongly supports the hypothesis that BotFinder is able to identify real malware infections with a relatively low number of false positives.

Whereas the 302 blacklist-confirmed IP addresses do not cluster to specific /24 or /16 networks, the 238 non-confirmed IP addresses show multiple large clusters and in total, 85 IPs contribute to the Top-5 networks. Table 7.2 lists the Top-5 organizations that were found in the list of not blacklisted destina-

[4]The RBLS `http://rbls.org/` service allows to analysis a large number of blacklists using a single query. "RFC-ignorant" listings are ignored.

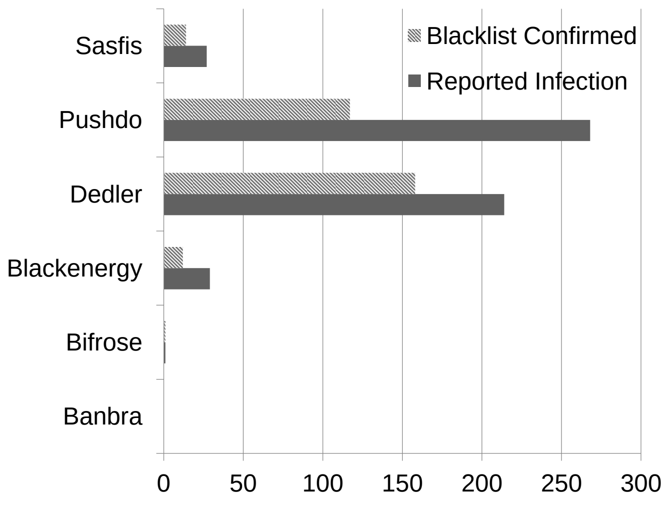


Figure 7.4: Total reported infections compared to the total number of blacklist confirmed infections.

tion IP addresses. 46 destination IP addresses point to servers of the Apple Incorporation and are most probably false positives. WebMagic[5] offers a variety of web services and WebNX[6] advertises dedicated web servers for rent. It is reasonable to speculate that malware authors rent dedicated servers and pay using maliciously obtained payment information to have access to powerful *C&C* servers while hiding their trails and evading law enforcement. However, this assumption is unable to be verified in this thesis. The Avtomatizatsiya Business Consulting Ltd, which is registered in Victoria, Seychelles offers rutracker.org, a BitTorrent tracker. Neither legitimate nor illegitimate activities were found.

If one adds Apple to the whitelist, effectively a blacklist-rate of 61% (302 of 496 destination IP addresses) is observed. Considering that various not blacklisted destination IP addresses belong to rented dedicated servers or other web providers, it is likely that a significant fraction of the 194 not blacklisted IP addresses actually belong to malicious servers.

[5]`http://www.webmagic.com/about.php`
[6]`http://webnx.com/`

Size	Network	Organization
46	17.0.0.0/8	Apple Inc.
21	67.207.64.0/19	WebMagic, Inc.
7	195.82.146.0/23	Avtomatizatsiya Business Consulting Ltd
6	216.18.192.0/19	WebNX, CA, USA
5	124.40.51.0/24	NTT / Akamai International BV, JP

Table 7.2: Top-5 aggregated clusters of not blacklisted destination IP addresses.

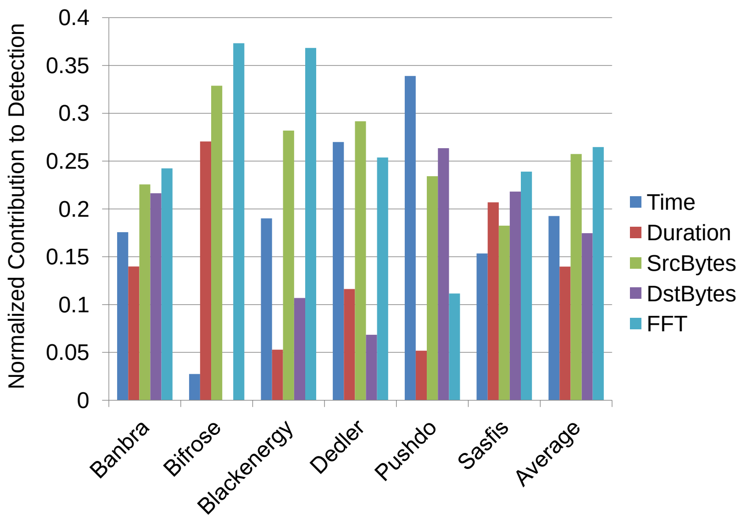


Figure 7.5: Normalized contribution of the different features towards a successful detection.

7.5 Contribution of Features towards Detection

To asses the quality of BotFinder's detection algorithm and the weighting of the different features towards a successful detection, the normalized contribution of each feature to γ_m is extracted. Figure 7.5 shows the averaged contribution of each feature to successful trace identification as the correct malware.

Interestingly, fundamentally different distributions for the bots under investigation are found: Whereas the bot families of Banbra and Sasfis are equally periodic in each dimension, the remaining bots show significant discrepancies between the features. For Pushdo, the duration and the FFT is of lower significance and the detection is primarily based on the average time interval and the number of bytes transmitted on average. The feature of destination bytes is of low importance for the remaining three bot families Bifrose, Blackenergy and Dedler, whereby Bifrose does not benefit from the feature at all. However, the source byte destination – the request towards the *C&C* server – highly contributes towards detection.

Of special interest is the impact of the Fast Fourier Transform, especially considering that the FFT accounts for the vast majority of the overall computational complexity of BotFinder. For all malware families except Dedler and Pushdo, the FFT is the the most significant feature to detect a malware infection in the network traffic. Hereby, Bifrose is of special interest, as the average time feature contributes only minimally towards detection whereas the FFT contributes most. This indicates a much better quality and periodicity of underlying frequencies compared to simple averaging – an indication that is verified under inspection of the underlying models which cluster significantly better for the FFT frequencies.

For the averaged contribution over all malware families (the rightmost bars), a relatively balanced contribution is observed. Still, the FFT and the source bytes dimension contribute slightly more towards successful detection than, e.g., the average duration or the destination bytes. Considering the mode of operation of typical bots, this outcome fits the assumption of bots to send similar requests to the *C&C* and receive answers of changing size. Additionally, the detection superiority of underlying FFT frequencies over simple averages for the flow interval times was shown.

Chapter 8

Bot Evolution

BotFinder is raising the bar for malware authors by detecting malware without relying on deep packet (content) inspection. Thereby, it might trigger a new round of bot evolution. This chapter presents potential evasion techniques that malware authors might try to thwart BotFinder and it discusses how the system handles such threats

8.1 Adding Randomness

A first approach to hinder BotFinder's analysis is to tackle the core assumption of regularity in the communication between bots and their *C&C* servers. For the bots under investigation, this assumption holds but malware authors might intentionally modify the communication patterns of their bots to evade detection, as suggested, for example, by Stinson et al. [69].

More specifically, botnet authors could randomize the time between connections from the bot to the *C&C* server or the number of bytes that are exchanged. For the botmaster, this comes at the price of loss of network agility and degraded information propagation within the botnet. However, by using randomization techniques, the malware author effectively increases the standard deviation for the property that is randomized (for example, the inter-trace timing). This decreases the quality of the trace for BotFinder, which in turn degrades the detection quality. Interestingly, BotFinder already operates on fluctuating traces and is, as the detection results show, robust against significant randomization (large standard deviations) around the average.

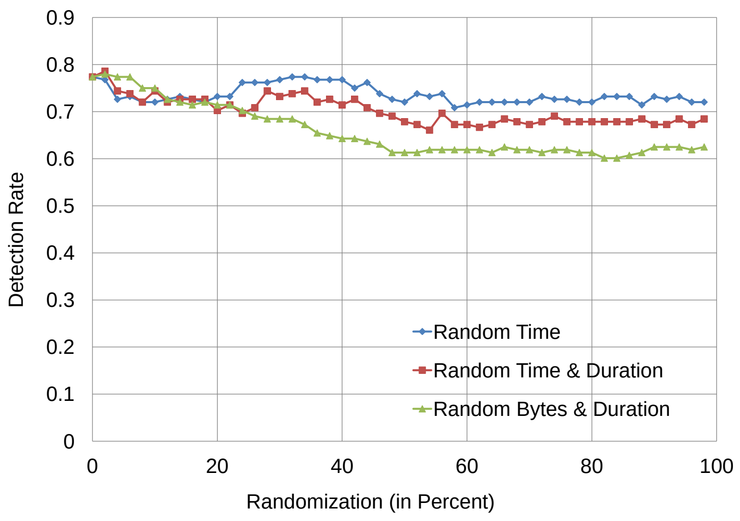


Figure 8.1: Randomization Impact.

To further illustrate BotFinder's resilience against randomization, the *C&C* trace detection rate with artificially increasing randomization is analyzed. In this experiment, a randomization of 50% means that 50% of the mean value is added or subtracted to each flow that composes a trace. For example, for a perfect trace with no standard deviation and an average interval of 100 seconds between connections, a randomization rate of 20% leads to new intervals between 80 and 120 seconds.

Figure 8.1 shows the effect on the detection rate of BotFinder with

1. randomized time, which impacts the "average time" and the "FFT" feature,
2. randomization of time and "duration", and
3. randomized "source bytes", "destination bytes," and "duration".

As can be seen, BotFinder's detection rate drops slightly but remains stable above 60% even when the randomization reaches 100%. However, for completeness one has to state that a randomization of all features quickly degrades the detection results. A 40% randomization of all dimensions degrades the detection rate to 35%.

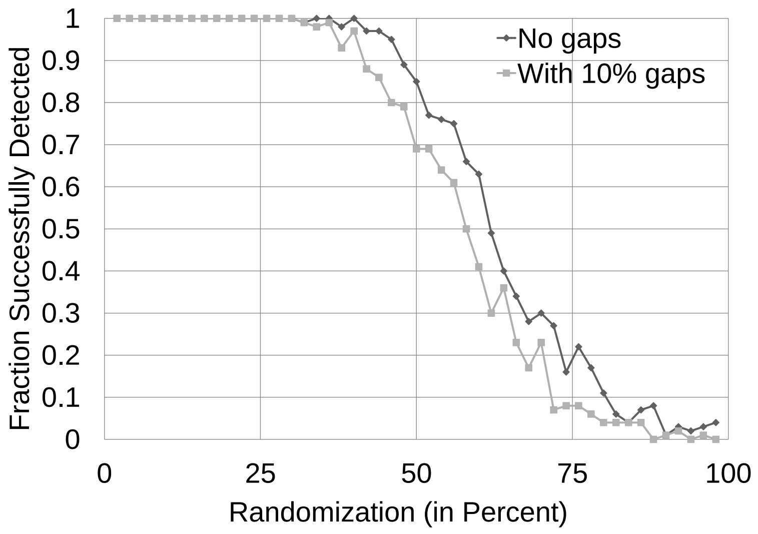
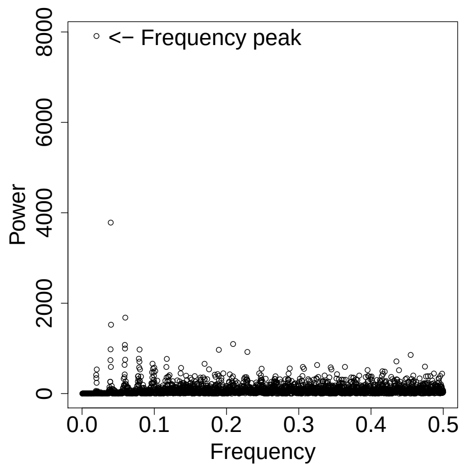


Figure 8.2: FFT detection quality degradation.

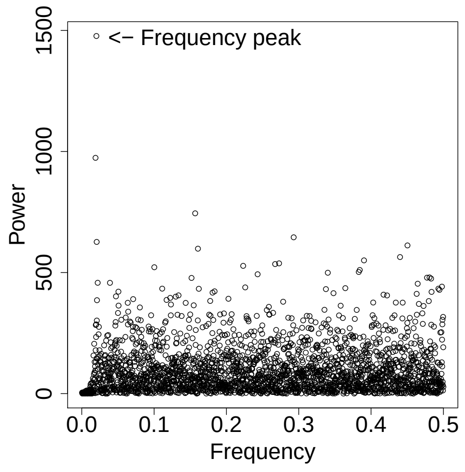
8.2 Introducing Larger Gaps

Malware authors might try to evade detection by adding longer intervals of inactivity between *C&C* connections. In this case, the Fast Fourier Transform significantly increases BotFinder's detection capabilities: Due to its ability to separate different *C&C* communication periodicities, the introduction of large gaps into the trace (which impacts the average) does not significantly reduce the FFT detection rate. For a randomization between 0 and 100 percent of the base frequency, Figure 8.2 shows the fraction of FFTs that detected the correct, underlying communication frequency. As can be seen, the introduction of large, randomly distributed long gaps does not significantly reduce the detection quality of the FFT-based models.

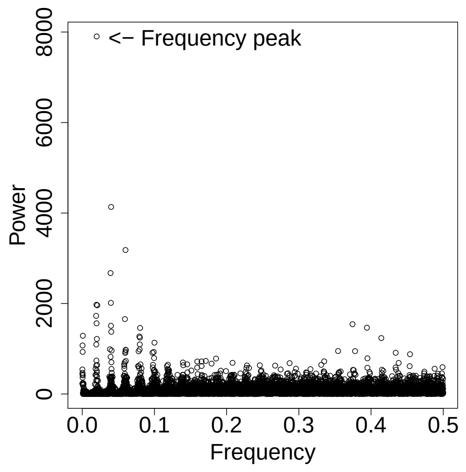
This is also illustrated for 10% and 50% randomization and the introduction of gaps in Figure 8.3. For a time interval of 50 minutes, the expected peak at $f = 0.02\frac{1}{min}$ is still visible.



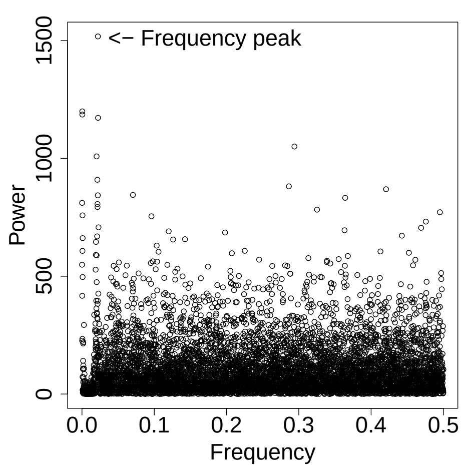
(a) 10% randomization, no gaps.



(b) 50% randomization, no gaps.



(c) 10% randomization, 10% long gaps.



(d) 50% randomization, 10% long gaps.

Figure 8.3: Frequency recognition using the FFT power spectrum. Despite increasing randomization [a) 10%, b) 50%] and additional ≈ 10 large gaps in c) and d), the correct time interval of 50 minutes (Frequency=0.02) is identified.

8.3 High Fluctuation of C&C Servers

Malware programs might try to exploit the fact that BotFinder requires to collect a certain, minimal amount of data for analysis. If the *C&C* server IP addresses are changed very frequently, BotFinder cannot build traces of minimal length $|\mathcal{T}|_{min} = 50$. This behavior of quickly changing the *C&C* server address or domain is referred to as IP-flux or domain flux respectively [54, 9]. Even a highly domain-fluxing malware, such as Torpig [71] uses two main communication intervals of 20 minutes (for upload of stolen data) and 2 hours for updating server information. Still, Torpig changes the *C&C* server domain in weekly intervals. *Currently, high C&C server fluctuations (IP flux) in the order of hours are not observed in the collected malware data investigated in this thesis.*

Nevertheless, the following pre-processing step already presents a countermeasure against highly fluctuating *C&C* servers. This step, which is based on elements of horizontal correlation, might finally also help to detect P2P based botnets. The pre-processing step is an optional step that operates before the full feature extraction (Step 4 in Figure 4.1). Hereby, longer traces are constructed from shorter traces, e.g, of length 20 to 49, which exhibit similar statistical features. In this step, the observation that *C&C* communication exhibits higher regularity than other frequent communication is used again.

8.3.1 Recombination

For all traces in a range of a new minimal trace length and $|\mathcal{T}_{min}| - 1$, which will be referred to as *sub-traces* in the following, a similarity definition analog to BotFinder's model matching algorithm is used. The decision to merge two sub-traces $\mathcal{T}_A$ and $\mathcal{T}_B$ is based on the following:

- To consider two traces similar, the standard deviation of the combined $\mathcal{T}_{AB}$ has to be lower than the standard deviation of at least one of the individual traces. Thereby, traces around a significantly different average – even with relatively low fluctuations – do not match and are automatically excluded.
- To rate each feature, a quality rating function $q = \exp\left(-\frac{\sigma}{avg}\right)$, with σ being the standard deviation of the specific feature of the trace, and avg

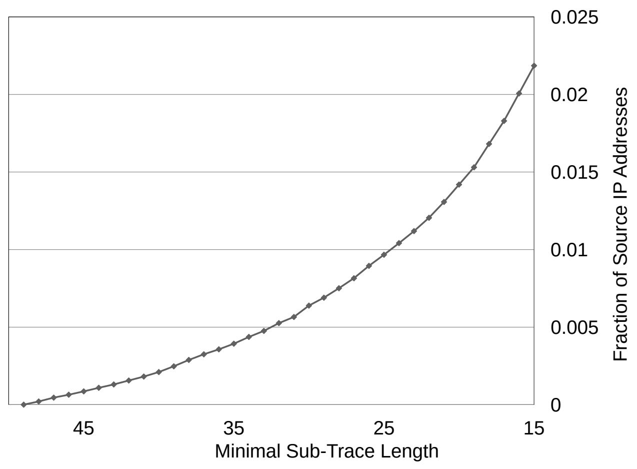


Figure 8.4: Fraction of source IPs with more than one sub-trace of length between the "Minimal Trace Length" and 49.

being the feature average, is used. If the sum over all feature-qualities of the combined trace $\mathcal{T}_{AB}$ is above a threshold t, the trace recombination is accepted.

The lower the value of t is, the more sub-trace combinations are accepted and the higher is the additional workload for BotFinder.

8.3.2 Sub-Trace Distribution

To judge the feasibility of the pre-processing step in terms of computational complexity and overhead, the amount of traces that require recombination and their typical, internal trace distribution needs to be known. For this purpose, 10 days of traffic in the *ISPNetFlow* dataset were analyzed and the fraction of source IPs that contributes to potential trace re-combination was extracted.

As can be seen in Figure 8.4, even considering a lower bound for the sub-trace length of 15 connections, only 2.1% of all IP addresses require the application of the pre-processing step in the 10 day timeframe.

Figure 8.5 shows the amount of sub-traces, that the different source IP addresses from Figure 8.4 have for a minimal sub-trace length of 15. More than 50% of all traces only have 2 sub-traces and more than 95% have less than 10

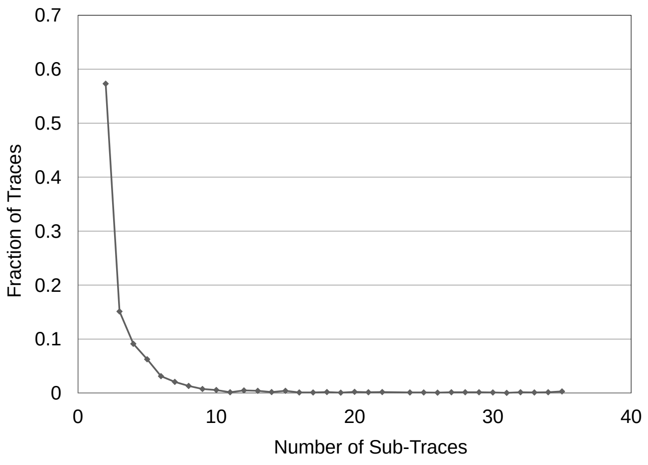


Figure 8.5: Fraction of IP addresses having exactly a given “Number of Sub-Traces”.

sub-traces. For the 10 day timeframe, only 58 IPs had 20 or more traces to recombine.

8.3.3 Computational Complexity

The computational complexity of each comparison of two traces for a fitting match is based on the calculation of the four features average time, average number of source bytes, average number of destination bytes and the average duration of flows. Therefore only 4 averaging operations and 4 standard deviation calculations are required for each matching operation. However, the *number of match-tests* is heavily influencing the run time of the pre-processing step.

The simplest approach is testing all possible trace combinations composed of two or more traces. For n sub-traces, f operations given by

$$f(n) = \sum_{k=2}^{n} \left(\frac{n!}{(n-k)!} \right) \tag{8.1}$$

are required. Table 8.1 illustrates the amount of possible trace combinations. Unfortunately, with $f = 3.5 \cdot 10^{12}$ combinations for only 15 sub-traces and a

Sub-Traces n	All Combinations	Chronologically Ordered	4-Trace Combinations
2	3	2	1
3	12	5	4
4	60	12	11
5	320	27	25
6	1,950	58	50
7	13,692	121	91
8	109,592	248	154
9	986,400	503	246
10	9,864,090	1,014	375
20	$6.6 \cdot 10^{18}$	1,048,556	6,175
30	$7.2 \cdot 10^{32}$	1,073,741,794	31,900

Table 8.1: Worst case amount of required combination steps using n sub-traces.

factorial run-time, this approach of analyzing all trace combinations is impossible to deploy in real world applications.

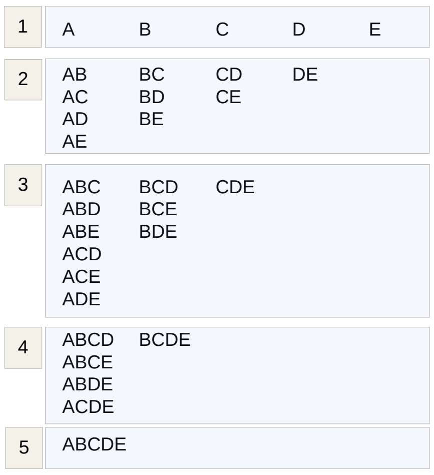
To optimize the run-time of the match making, one can leverage the property of two traces to commutate, meaning $\mathcal{T}_A\mathcal{T}_B$ is equivalent to $\mathcal{T}_B\mathcal{T}_A$ in its statistical properties. Please note, that this effect becomes of significance for longer combinations ($\mathcal{T}_A\mathcal{T}_B\mathcal{T}_C$=$\mathcal{T}_A\mathcal{T}_C\mathcal{T}_B$= $\mathcal{T}_B\mathcal{T}_A\mathcal{T}_C$=$\mathcal{T}_B\mathcal{T}_C\mathcal{T}_A$=$\mathcal{T}_C\mathcal{T}_A\mathcal{T}_B$= $\mathcal{T}_C\mathcal{T}_B\mathcal{T}_A$).

Effectively, it is not required to investigate the commutations: Sub-traces contain chronological information and can be sorted accordingly. Therefore, $\mathcal{T}_A$ and $\mathcal{T}_B$ are resolved to one sequence by concatenation in chronological order. As Figure 8.6 illustrates for five sub-traces, each sub-trace allows combinations with sub-traces that are “ordered behind”, e.g., by starting later in time. The number of comparison operations can therefore directly be expressed as:

$$f(n) = 2^n - n \tag{8.2}$$

The amount of individual traces are of no relevance (because each trace length is $< |\mathcal{T}|_{min}$) leading to the substraction of n in Equation 8.2. Although this modification significantly reduces the amount of possible trace combinations as shown in Table 8.1, further optimization has to be done to limit negative performance impacts for larger amounts of sub-traces.

Following the original assumption to require a minimal sub-trace length of



1	A	B	C	D	E
2	AB AC AD AE	BC BD BE	CD CE	DE	
3	ABC ABD ABE ACD ACE ADE	BCD BCE BDE	CDE		
4	ABCD ABCE ABDE ACDE	BCDE			
5	ABCDE				

Figure 8.6: Chronologically ordered potential combinations of five sub-traces A to E.

15 and a full trace length of 50, it is justified to abort the recombination procedure after combining a maximum of 4 sub-traces. The number of traces that are combined of two sub-traces is given by:

$$f_2(n) = \sum_{i=1}^{n}(n-i) = \frac{1}{2}(n^2+n+i^2-i)-ni \tag{8.3}$$

The number of traces combined from 3 sub-traces is given by:

$$f_3(n) = \sum_{j=2}^{n}\sum_{i=j}^{n}(n-i) \tag{8.4}$$

and the number of traces combined from 4 sub-traces by:

$$f_4(n) = \sum_{k=3}^{n}\sum_{j=k}^{n}\sum_{i=j}^{n}(n-i) \tag{8.5}$$

The sum of the sub-traces combined from 2, 3 and 4 sub-traces results to:

$$f_{2..4}(n) = f_2(n)+f_3(n)+f_4(n) = \frac{1}{24}n^4 - \frac{2}{24}n^3 + \frac{11}{24}n^2 - \frac{10}{24}n \tag{8.6}$$

This leads to a low amount of match analysis operations and generated traces compared to the previous solutions. By considering the ordering of traces and a targeted trace composed of two to four sub-traces, the complexity is reduced from factorial complexity over exponential to polynomial complexity. Nevertheless, the final algorithm still has – in the worst case that every trace matches with every other – a complexity scaling in $\mathcal{O}(n^4)$. Still, for a trace that has 50 highly similar sub-traces, $\approx 250,000$ new traces are generated.

As the re-combinations are performed without FFT or other costly operations and just build around simple summation, averaging and standard deviation calculation, the processing is sufficiently fast to handle even larger amounts of sub-traces. Please note, that in a typical scenario, most sub-traces will not be similar enough to be combined and the actual amount of operations is significantly lower.

8.3.4 Real World Impact

The pre-processing step was applied on real-world data to investigate

1. the ability to reassemble real *C&C* traces,
2. the reassembly difference between real *C&C* and other, long traces,
3. the amount of additional traces that actually need to be analyzed by BotFinder and the implied additional workload, and
4. the false positive ratio of the newly generated traces.

Figure 8.7 illustrates the reassembly rates for bisected real *C&C* traces (strict purification on the six malware families) compared to the the reassembly rate of long non-*C&C* traces from the *LabCapture* dataset. The acceptance threshold t is varied and for $t = 1.9$ BotFinder *reassembled 91% of the real C&C traces* and combined only 8% of non-*C&C* long traces.

Furthermore, when running on the 2.5 months of *LabCapture* data, the system reassembled 3.4 million new traces. Using the same detection threshold as in the evaluation ($\alpha = 1.8$), these traces *do not introduce any new false positives.* In general, it can be observed that the false positive fraction of reassembled traces is lower than the original false-positive fraction for normal long traces.

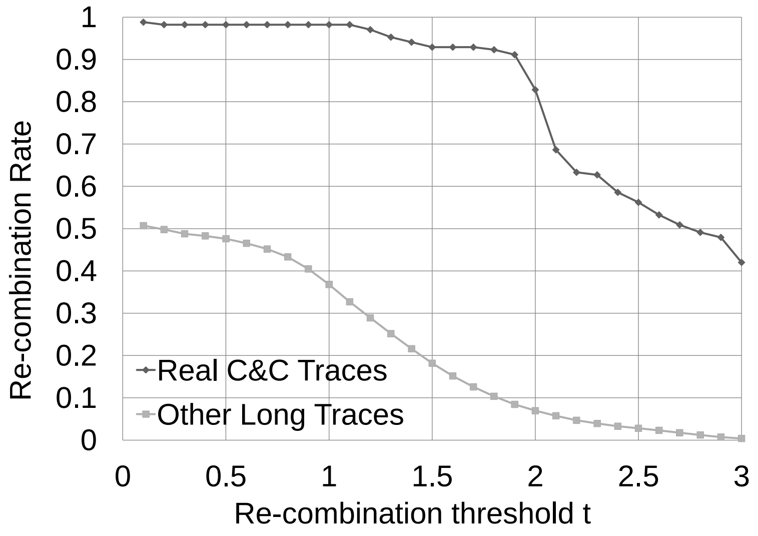


Figure 8.7: The recombination rate using different thresholds t during pre-processing.

As pre-processing is targeted to be run over shorter time frames – as it is a countermeasure against IP fast flux – even fewer new traces will be generated. For example, in a ten day NetFlow traffic set, only 0.6% of the IP addresses with more than one sub-trace generated additional traces. Computing these traces increased the workload for BotFinder by 85% compared to normal operation. Thus, with a modest increase in overhead, BotFinder also covers cases where bots frequently switch IP addresses.

8.4 P2P Bots

BotFinder might be able to detect P2P networks by concatenating the communication to different peers in one trace. Nevertheless, complementing BotFinder with elements from different existing approaches might be beneficial: BotFinder could, for example, be expanded by a component that creates structural behavior graphs, as proposed by Gu et al. [31, 30], or be complemented by P2P net analysis techniques similar to BotTrack [24] or BotGrep [52], which try to reveal members of a bot network by surveillance of a single member of the network. Still, completely changing to a P2P-based botnet also imposes significant challenges for the botmaster. These include the ease of enumeration of all participating bots by every member in the botnet (for example, a honeypot-caught bot under control of a security researcher as performed by BotGrep), and the time to disseminate commands. Hence, most botnets today use a centralized infrastructure.

8.5 Bot-like Benign Traffic

Although unlikely, benign communication might accidentally exhibit a similar traffic pattern as a bot family. For example, a POP3 mail server might get queried in the same interval as a bot communicates with its *C&C* server, and the traffic sizes might accidentally match. If these services operate on a static IP, a system administrator can easily exclude these false positives by whitelisting this IP address. A local BotFinder installation should be configured to ignore communication between hosts under the same local authority. For popular web services with similar features, a generic whitelisting is possible.

8.6 Discussion

BotFinder is able to learn new communication patterns during training and is robust against the addition of randomized traffic or large gaps. Furthermore, given the pre-processing step, even changing the *C&C* server frequently is highly likely to be detected. Nevertheless, BotFinder is completely reliant on statistical data and regularities. If the attacker is willing to:

1. significantly randomize the bot's communication pattern, and
2. drastically increase the communication intervals to force BotFinder to capture traces over longer periods of time, and
3. introduce overhead traffic for source and destination byte variation, and
4. change the *C&C* server extremely frequent, e.g., after each tenth communication, and
5. use completely different traffic patterns after each *C&C* server change, then

BotFinder's detection fails as minimal or no statistical consistency can be found anymore. On the contrary, a malware author who implements such evasion techniques, has to trade the botnets performance in order to evade BotFinder: Using randomization and additional traffic increases the overhead and reduces synchronization and the network-agility of the botnet. In particular, especially the frequent change of *C&C* servers is costly and requires an increased amount of work and cost by the botmaster: Domains need to be pre-registered and paid and new globally routeable IP addresses must be obtained. Hereby, the bots need to know to which *C&C* server to connect, so the new domains must either follow a pre-defined and malware-hardcoded pattern – which allows take-over attacks by security researchers such as in Stone-Gross et al. [71] (with a weekly changing domain) – or lists of new *C&C* servers need to be distributed to the members of the botnet. Both ways increase the botnet operator's costs and reduce stability and performance of the malware network.

Chapter 9

Related Work

Research in bot detection using network traffic analysis can be classified into two main directions as depicted in Figure 9.1: The first direction is that of *vertical correlation*, in which network events and traffic are inspected for typical evidences of bot infections such as scanning, *C&C* communication, or denial of service attacks. A well known representative of vertical correlation is *Bothunter* [31], which heavily relies on a modified Snort[1] and uses a combination of signature and anomaly-based intrusion detection components. In detail, *Bothunter* leverages detection mechanisms on the whole infection and malware execution life-cycle: Port scanning activities and potentially dangerous binary transfers (e.g., encoded or encrypted HTTP POSTs or shell code) are used to detect a first step of the infection process. Malware loads ("egg downloads") and, finally, structural information regarding the *C&C* server plus IP blacklisting of multiple list providers are used to identify infected hosts. *Bothunter* finally uses a threshold metric based on IP destinations, blacklisting and the observed behavior to raise alarms and classify attacks. More classical vertical approaches are employed by Goebel et al. [28] (Rishi) and Binkley et al. [7] which examine and model IRC-based network traffic for nickname patterns that are frequently used by bots. Karasaridis et. al [38] detects IRC bots using fixed controller ports and flow information. Unfortunately, these techniques are tailored to a specific botnet structure [28, 7] or rely on the presence of a specific bot-infection life-cycle [31]. Moreover, these techniques rely on the presence of noisy behavior such as scan, spam, or DoS traffic.

Wurzinger et al. [80] and Perdisci et al. [58] automatically generated signatures

[1]`http://www.snort.org`

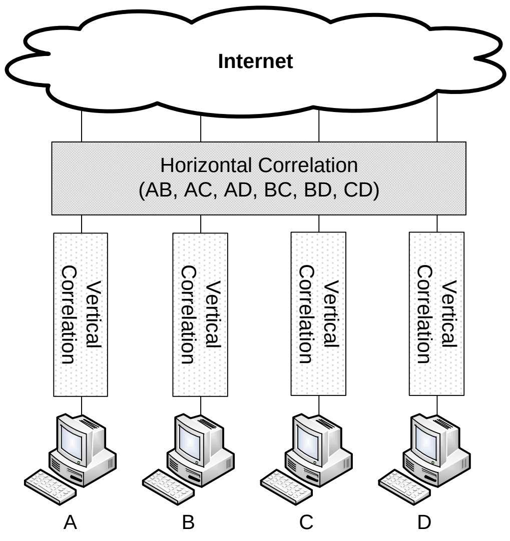


Figure 9.1: Vertical and horizontal correlation for malware detection.

that represent the behavior of an infected host. The key point in both strategies is that bots receive commands from the bot master and then respond in specific ways. The authors show that it is possible to identify bots' responses and use this information to encode a network signature that can be plugged into a NIDS. The approaches are very interesting and promising, showing a very high detection rate and a limited false positives ratio. Unfortunately, both techniques require to inspect packet content and can thus be circumvented by encrypting the *C&C* communication. Giroire et al. [27] presented an approach to detect *C&C* communications by looking for temporal relationships in the connections of an end-host. This approach is similar to BotFinder as both focus on temporal relationships found in communication patterns. However, BotFinder differs fundamentally in the way malware detection is performed. In particular, [27] is based on the concept of destination atoms and persistence. Destination atoms group together communications towards a common service or web-address, whereas the persistence is a multi-granular measure of destination atoms' temporal regularity. The idea consists in observing the per-host initiated connections for a certain (training) period and grouping them into destination atoms. Subsequently, very persistent destination atoms, i.e., those whose persistence level is above a fixed threshold, form a host's whitelist, which will be compared against the very persistent destination atoms found once the training session ends. According to the authors, the resulting whitelist is reasonably small as the majority of destination atoms a host is engaged with are transient, i.e., non-persistent. Thus, very persistent destination atoms will be flagged as anomalous and potentially identify a *C&C* host.

The second direction is the *horizontal correlation* of network events from two or more hosts, which are involved in similar, malicious communication. Interesting approaches are represented by BotSniffer [32], BotMiner [30], TAMD [82], and the work by Strayer et al. [72]. Except the latter, which works on IRC analysis, the main strength of these systems is their independence of the underlying botnet structure, and thus, they have shown to be effective in detecting pull-based, push-based, and P2P-based botnets. By contrast, correlating actions performed by different hosts requires that at least two hosts in the monitored network are infected by the same bot and that the bot behavior exhibits characteristics that are statistically significant enough to correlate to. As a consequence of the requirement to have multiple infections in the network, these techniques cannot detect single bot-infected hosts, which is a significant limitation, especially considering the trend toward smaller botnets [16]. In addition, the detection mechanisms are usually triggered once malicious and noisy behavior, such as scan, spam, and DDoS, is observed [30]. For BotMiner, malware detection is performed by correlating events from a flow level control plane (with traces similar to BotFinder) with malicious activities observed in the so-called activity plane. Effectively, only malware that is already detected via vertical correlation is considered in the control plane analysis. This reliance on noisy behavior significantly reduces the advertised zero-day detection ability and detection of stealthy bots. Moreover, low-pace, non-noisy, and profit-driven behavior [23, 33] is getting predominant in current bots as witnessed in the past few years [71].

A way to detect P2P botnets is shown in BotGrep [52] and BotTrack [24], which leverage the underlying communication infrastructure in the P2P overlay. Whereas BotGrep uses specifics of the DHT interactions, BotTrack operates on NetFlows only and is comparable to BotFinder in this aspect. However, BotGrep and BotTrack need to be bootstraped with the botnet under investigation, typically by utilizing a participating active malware sample in a honeypot. Connections of this bot under surveillance reveal other members of the network. This requirement of an active source in the honeypot is a significant drawback. Nevertheless, concepts from these solutions might complement BotFinder to allow detection of P2P based bots during NetFlow analysis as well.

Chapter 10

Conclusion

This thesis demonstrated that bots – malware that is remotely controlled by a *C&C* server – exhibit a network communication pattern, which significantly differs from normal, benign traffic. This difference manifests in each of four different dimensions of traffic that were analyzed. These dimensions are the time interval between connections to a server, the number of bytes transferred to and from the server, and the duration of connections. For each dimension, a strong periodicity in bot traffic was observed, which distinguishes it from benign traffic.

Based on this difference in network traffic, a malware detection system called BotFinder was created. BotFinder utilizes five features – averages over the aforementioned four dimensions and an additional Fast Fourier Transform over the time intervals – to capture the statistical properties of a *C&C* communication. To this end, traces of recurring communication between two IP addresses on the same destination port are created.

BotFinder uses machine learning to train on malware traffic that is obtained by executing malicious software in a controlled sandbox environment. The result of this training step are bot-models consisting of clustered features of the communication, for example the time interval between connections. The malware families clustered quite well, meaning typically around three clusters per feature with low relative standard deviation. These models are finally applied to network traffic to detect potential malware infections *without the need of deep packet inspection*. Especially the latter is a significant improvement over related work, as it reacts on the trend of malware authors to encrypt and stealth the *C&C* communication. Moreover, deployment is simplified and privacy concerns are mitigated as end-user's traffic is not inspected anymore.

The evaluation of BotFinder showed that it outperforms the content inspection based system *Bothunter* and has a high detection rate of 77% as well as relatively low false positives. Applied to a dataset of a large ISP, BotFinder indicated 542 end hosts to be infected with a bot. For 56% of the destination IP addresses, entries were found in publicly available blacklists. A cluster analysis of the remaining non-blacklisted IP addresses revealed that a large cluster connected to Apple, which should be added to the BotFinder whitelist, and other clusters connected to websites that might host malicious *C&C* servers.

With the additional means to counteract potential detection evasion strategies, such as the introduction of randomness or larger gaps in the communication, BotFinder is a robust malware detection system that effectively complements deployed end-host AV scanners.

Overall, BotFinder proved that the statistical anomalies investigated in this thesis are sufficient to perform content agnostic network level malware detection with high detection rates and low false positives. Thereby, this thesis effectively raises the bar for malware authors and potentially lowers the efficiency of future botnets.

Bibliography

[1] C. Akass. Storm worm making millions a day http://www.computeractive.co.uk/pcw/news/1923144/storm-worm-millions-day.

[2] N. Archak, A. Ghose, and P. G. Ipeirotis. Show me the money!: deriving the pricing power of product features by mining consumer reviews. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 56–65, New York, NY, USA, 2007. ACM.

[3] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. 2009.

[4] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR)*, April 2006.

[5] U. Bayer, C. Kruegel, and E. Kirda. Anubis: Analyzing Unknown Binaries. In *http://anubis.iseclab.org/*, 2008.

[6] F. Bellard. Qemu, a Fast and Portable Dynamic Translator. In *Usenix Annual Technical Conference*, 2005.

[7] J. R. Binkley. An algorithm for anomaly-based botnet detection. In *SRUTI '06*, pages 43–48, 2006.

[8] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring pay-per-install: the commoditization of malware distribution. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[9] A. Caglayan, M. Toothaker, D. Drapaeau, D. Burke, and G. Eaton. Behavioral analysis of fast flux service networks. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, CSIIRW '09, pages 48:1–48:4, New York, NY, USA, 2009. ACM.

[10] R. B. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3:1–27, 1974.

[11] F. Chang, W. Qiu, R. H. Zamar, R. Lazarus, and X. Wang. clues: An r package for nonparametric clustering based on local shrinking. *Journal of Statistical Software*, 33(4):1–16, 2 2010.

[12] K. Chiang and L. Lloyd. A Case Study of the Rustock Rootkit and SPAM Bot. In *HotBots'07: Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, pages 10–10, Berkeley, CA, USA, 2007. USENIX Association.

[13] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), Oct. 2004.

[14] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), Jan. 2008.

[15] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[16] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: understanding, detecting, and disrupting botnets. In *SRUTI'05: Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet*, pages 6–6, Berkeley, CA, USA, 2005. USENIX Association.

[17] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. volume 19, pages 297–301. American Mathematical Society, 1965.

[18] D. Dagon, G. Gu, C. Lee, and W. Lee. A taxonomy of botnet structures. In *Proceedings of the 23 Annual Computer Security Applications Conference (ACSAC'07)*, December 2007.

[19] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 51–62, New York, NY, USA, 2008. ACM.

[20] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th conference on USENIX Security Symposium*, volume 13 of *SSYM'04*, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[21] P. Fogla and W. Lee. Evading network anomaly detection systems: formal reasoning and practical techniques. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 59–68, New York, NY, USA, 2006. ACM.

[22] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, Berkeley, CA, USA, 2006. USENIX Association.

[23] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[24] J. François, S. Wang, R. State, and T. Engel. Bottrack: Tracking botnets using netflow and pagerank. In J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, editors, *NETWORKING 2011*, volume 6640 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg, 2011.

[25] F. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In S. di Vimercati, P. Syverson, and D. Gollmann, editors, *Computer Security ESORICS 2005*, volume 3679 of *Lecture Notes in Computer Science*, pages 319–335. Springer Berlin / Heidelberg, 2005.

[26] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32 – 40, jan 1975.

[27] F. Giroire, J. Chandrashekar, N. Taft, E. M. Schooler, and D. Papagiannaki. Exploiting Temporal Persistence to Detect Covert Botnet Channels. In *RAID*, pages 326–345, 2009.

[28] J. Goebel and T. Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *HotBots'07: Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, pages 8–8, Berkeley, CA, USA, 2007. USENIX Association.

[29] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 27–37, New York, NY, USA, 2010. ACM.

[30] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[31] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)*, August 2007.

[32] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[33] P. Gutmann. The Commercial Malware Industry. In *Proceedings of the DEFCON conference*, 2007.

[34] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108, 1979.

[35] Ipsos Public Affairs. MAAWG Email Security Awareness and Usage Survey, 2010.

[36] G. Jacob, R. Hund, C. Kruegel, and T. Holz. Jackstraws: Picking Command and Control Connections from Bot Traffic. *Usenix Security Symposium*, August 2011.

[37] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: an empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, CCS '08, pages 3–14, New York, NY, USA, 2008. ACM.

[38] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale Botnet Detection and Characterization. In *HotBots'07: Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, pages 7–7, Berkeley, CA, USA, 2007. USENIX Association.

[39] L. Kaufman and P. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis.* Wiley Interscience, New York, 1990.

[40] D. M. R. Kernighan. *The C Programming Language 2nd ed.* Englewood Cliffs, NJ: Prentice Hall, 1988.

[41] N. Kshetri. The economics of click fraud. volume 8, pages 45–53, Piscataway, NJ, USA, May 2010. IEEE Educational Activities Department.

[42] S. Kundu. Gravitational clustering: a new approach based on the spatial distribution of the points. *Pattern Recognition*, 32(7):1149 – 1160, 1999.

[43] T. D. Lane. *Machine learning techniques for the computer security domain of anomaly detection.* PhD thesis, Purdue University, 2000.

[44] K. Levchenko, A. Pitsillidis, N. Chachra, B. Enright, M. Félegyházi, C. Grier, T. Halvorson, C. Kanich, C. Kreibich, H. Liu, D. McCoy, N. Weaver, V. Paxson, G. M. Voelker, and S. Savage. Click Trajectories: End-to-End Analysis of the Spam Value Chain. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 431–446, Washington, DC, USA, 2011. IEEE Computer Society.

[45] F. Li and M.-H. Hsieh. An empirical study of clustering behavior of spammers and group-based anti-spam strategies. In *CEAS 2006*, 2006.

[46] Z. Li, G. Xia, H. Gao, Y. Tang, Y. Chen, B. Liu, J. Jiang, and Y. Lv. Netshield: massive semantics-based vulnerability signature matching for high-speed networks. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 279–290, New York, NY, USA, 2010. ACM.

[47] Y. P. Mack and M. Rosenblatt. Multivariate k-nearest neighbor density estimates. *Journal of Multivariate Analysis*, 9(1):1 – 15, 1979.

[48] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[49] S. Malinin. Spammers earn millions and cause damages of billions http://english.pravda.ru/russia/economics/15-09-2005/8908-spam-0/, 2005.

[50] M. A. Maloof. *Machine Learning and Data Mining for Computer Security.* Springer-Verlag London, 2006.

[51] T. M. Mitchell. *Machine Learning.* McGraw-Hill Book Co, 1997.

[52] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. Botgrep: finding p2p bots with structured graph analysis. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, pages 7–7, Berkeley, CA, USA, 2010. USENIX Association.

[53] J. Nazario. BlackEnergy DDoS Bot Analysis. Technical report, Arbor Networks, 2007.

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

[55] Y. Niu, Y.-M, Wang, H. Chen, M. Ma, and F. Hsu. A Quantitative Study of Forum Spamming Using Contextbased Analysis. In *14th NDSS, 2007*, 2007.

[56] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928.

[57] R. Perdisci, D. Dagon, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *In Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 17–31, 2006.

[58] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *Proceedings of the 7th USENIX conference on Networked systems design*

and implementation, NSDI'10, pages 26–26, Berkeley, CA, USA, 2010. USENIX Association.

[59] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer. Information Model for IP Flow Information Export. RFC 5102 (Proposed Standard), Jan. 2008. Updated by RFC 6313.

[60] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[61] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 41–52. ACM Press, 2006.

[62] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 342–351, New York, NY, USA, 2007. ACM.

[63] D. Samosseiko. The Partnerka – What is it, and why should you care? In *Virus Bulletin Conference*, 2009.

[64] Y. Shin, M. Gupta, and S. Myers. Prevalence and Mitigation of Forum Spamming. In *INFOCOM*, 2011.

[65] F. Silveira, C. Diot, N. Taft, and R. Govindan. Astute: detecting a different class of traffic anomalies. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 267–278, New York, NY, USA, 2010. ACM.

[66] R. C. Singleton. On computing the fast fourier transform. *Commun. ACM*, 10(10):647–654, Oct. 1967.

[67] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. In R. Sekar and A. Pujari, editors, *Information Systems Security*, volume 5352 of *Lecture Notes in Computer Science*, pages 1–25. Springer Berlin / Heidelberg, 2008.

[68] J. Stewart. BlackEnergy Version 2 Analysis. Technical report, SecureWorks' Counter Threat Unit (CTU), 2010.

[69] E. Stinson and J. C. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *USENIX WOOT*, 2008.

[70] B. Stone-Gross, R. Abman, R. Kemmerer, C. Kruegel, D. Steigerwald, and G. Vigna. The Underground Economy of Fake Antivirus Software. In *10th Workshop on the Economics of Information Security (WEIS).*, June 2010.

[71] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM.

[72] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting botnets with tight command and control. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pages 195–202, 2006.

[73] R. S. Sutton and A. G. Barto. *Reinforcement Learning - An Introduction.* MIT Press, Cambridge, 1998.

[74] B. Trammell and E. Boschi. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103 (Proposed Standard), Jan. 2008.

[75] Trusteer. Measuring the in-the-wild effectiveness of antivirus against zeus, September 2009.

[76] G. van Rossum. Python Tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

[77] X. Wang, W. Qiu, and R. H. Zamar. Clues: A non-parametric clustering method based on local shrinking. *Computational Statistics & Data Analysis*, 52(1):286 – 298, 2007.

[78] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, 5:32–39, 2007.

[79] W. Wright. Gravitational clustering. *Pattern Recognition*, 9(3):151 – 166, 1977.

[80] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically Generating Models for Botnet Detection. In *14th European Symposium on Research in Computer Security (ESORICS), Lecture Notes in Computer Science, Springer Verlag*, September 2009.

[81] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming botnets: signatures and characteristics. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 171–182, New York, NY, USA, 2008. ACM.

[82] T.-F. Yen and M. K. Reiter. Traffic Aggregation for Malware Detection. In *DIMVA '08: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 207–227, Berlin, Heidelberg, 2008. Springer-Verlag.

Curriculum Vitae

Personal Data

Florian Tegeler
An der Bleichwiese 20
37073 Göttingen, Germany

Phone: +49 551 2829341
E-Mail: florian.tegeler@gmail.com

Date of Birth: May 11th, 1982
Martial Status: Married
Nationality: German

Research Interests

Network Security

Malware Analysis and Detection

Social Networks

Education

since 08/2008	Student of the PCS PhD program at the University of Göttingen
07/2008	Diplom-Physiker, Physics, University of Göttingen, Grade: “sehr gut”
	Diploma Thesis: Dynamic Protonation of Titrable Groups in Biomolecules for Molecular Dynamics Simulations, Grade: 1.3

	Supervisors and Mentor: Prof. Dr. Helmut Grubmüller, Max Planck Institute for Biophysical Chemistry, Göttingen Prof. Dr. Annette Zippelius, University of Göttingen Dr. Gerrit Grönhof, Max Planck Institute for Biophysical Chemistry, Göttingen
02/2008	Master of Science, Computer Science, University of Göttingen, Grade: “sehr gut” Master Thesis: Security Analysis, Prototype Implementation and Performance Evaluation of a New IPSec Session Resumption Method, Grade: 1.0 Supervisors and Mentor: Prof. Dr. Xiaoming Fu, University of Göttingen Dr. Henrik Brosenne, University of Göttingen Hannes Tschofenig, Siemens
2002 - 2006	Computer science studies towards Bachelor of Science, University of Göttingen
2002 - 2004	Physics studies towards Intermediate Diploma, University of Göttingen

Publications

GEMSTONE: Empowering Decentralized Social Networking with High Data Availability, Florian Tegeler, David Koll, and Xiaoming Fu, IEEE GLOBECOM 2011 - Selected Areas in Communications Symposium - Social Networks Track, Houston, TX, USA, IEEE, December 2011.

Constant pH Molecular Dynamics in Explicit Solvent with lambda-Dynamics, Serena Donnini, Florian Tegeler, Gerrit Groenhof, and Helmut Grubmüller, Journal of Chemical Theory and Computation 7: 1962-1978, April 2011.

GEMSTONE: A Generic Middleware for Social Networks, David Koll, Florian Tegeler, and Xiaoming Fu, ACM/USENIX MobiSys 2010 poster session, ACM, June 2010.

SybilConf: Computational Puzzles for Confining Sybil Attacks, Florian Tegeler, and Xiaoming Fu, IEEE INFOCOM 2010 Student Workshop, March 2010.

Interest based automated content exchange in 7DS, Florian Tegeler, Technical Report IFI-TB-2009-02, Institute of Computer Science, University of Göttingen, ISSN 1611-1044, August 2009.

Security Analysis of IKEv2 Session Resumption, Florian Tegeler, Technical Report No. IFI-TB-2009-01, Institute of Computer Science, University of Göttingen, ISSN 1611-1044, June 2009.

A Unified Security Backplane for Trust and Reputation Systems in Decentralized Networks, Florian Tegeler, Jun Lei, and Xiaoming Fu, IEEE INFOCOM 2009 Student Workshop, April 2009.

Security Analysis, Prototype Implementation and Performance Evaluation of a New IPSec Session Resumption Method, Florian Tegeler, Zentrum für Informatik, University of Göttingen, Master's Thesis, No. ZFI-BM-2008-01, ISSN 1612-6793, January 2008.

Formal Specification and Security Verification of the IDKE Protocol using FDR Model Checking, Rene Soltwisch, Florian Tegeler, and Dieter Hogrefe, Proceedings of the 13th IEEE International Conference on Networks (ICON), IEEE, ISBN 1-4244-0000-7, November 2005.

Review of CasperFDR Analysis of the IDKE Protocol, Florian Tegeler, and Rene Soltwisch, Technical Report No. IFI-TB-2005-04, Institute of Computer Science, University of Göttingen, Germany, ISSN 1611-1044, June 2005.

Research Visits

11/2011	DAAD Exhange with Fudan University, Shanghai, China Host: Prof. Jin Zhao
03/2011-08/2011	University of California, Santa Barbara, CA, USA Host: Prof. Christopher Krügel

03/2009-07/2009	DAAD Exchange with Columbia University, NY, USA Host: Prof. Henning Schulzrinne
07/2006-10/2006	University of Cambridge, Cambridge, UK Host: Prof. Bill Clegg

Göttingen, April 10th, 2012