FINGERPRINTING MALICIOUS IP TRAFFIC

NOUR-EDDINE LAKHDARI

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

MARCH 2014

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Nour-Eddine Lakhdari**

Entitled: **Fingerprinting Malicious IP Traffic**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Chadi Assi _____ Chair

Dr. Lingyu Wang _____ Examiner

Dr. Wahab Hamou-Lhadj _____ External (to program)

Prof. Mourad Debbabi _____ Supervisor

Approved _____

Prof. Rachida Dssouli, Director

Concordia Institute for Information Systems Engineering

_____ 20 _____ _____

Prof. Christopher Trueman, Dean

Faculty of Engineering and Computer Science

Abstract

Fingerprinting Malicious IP Traffic

Nour-Eddine Lakhdari

In the new global economy, cyber-attacks have become a central issue. The detection, mitigation and attribution of such cyber-attacks require efficient and practical techniques to fingerprint malicious IP traffic. By fingerprinting, we refer to: (1) the detection of malicious network flows and, (2) the attribution of the detected flows to malware families that generate them. In this thesis, we firstly address the detection problem and solve it by using a classification technique. The latter uses features that exploit only high-level properties of traffic flows and therefore does not rely on deep packet inspection. As such, our technique is effective even in the presence of encrypted traffic. Secondly, whenever a malicious flow is detected, we propose another technique to attribute such a flow to the malware family that generated it. The attribution technique is built upon k-means clustering, sequence mining and Pushdown Automata (PDAs) to capture the network behaviors of malware family groups. Indeed, the generated PDAs are actually network signatures for malware family groups. Our results show that the proposed malicious detection and attribution techniques achieve high accuracy with low false (positive and negative) alerts.

Acknowledgments

I would like to express my special appreciation and thanks to my supervisor *Professor. Mourad Debbabi*. You have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless. Without his support, this thesis would not have been possible. Furthermore, I would like to thank the thesis committee: *Dr. Lingyu Wang*, *Dr. W. Hamou Lhadj* and *Dr. Chadi Assi* for their insightful, valuable and detailed comments. I would like to acknowledge the help of the *Phd. Amine Boukhtouta* at different stages of my research.

A special thanks to my family. Words cannot express how grateful I am to my Mothers: *Wahiba*, *Halima*, Father: *Salim*, *Moubarek*, my Brothers: *Ali*, *Amine*, *Abdellatif*, *Hebri* and Sisters: *Amina*, *Fatima*, *Malika* for all of the sacrifices that you have made on my behalf. Your prayer for me was what sustained me thus far. I would also like to thank all of my Friends: *Abderraouf Kafi*, *Riad Kafi*, *Mounir Chekhchoukh* and *Salim Chekhchoukh* who supported me in writing, and incented me to strive towards my goal.

At the end I would like express appreciation to my special Brother *Moubarek* the one that I believe will be better than the one that have been left from this world that share the same name.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

The propagation of malware and the increase of attacks in the cyberspace have urged organizations to elaborate security strategies that aim to detect and prevent cyber-attacks against networked infrastructures, decrease vulnerabilities and confine damages. However, the frequent occurrence of cyber-incidents and the various attack techniques make achieving cyberspace security objectives a much harder problem.

One of the major phenomena is the growing of Hacktivism [55]. Indeed, Anonymous has perpetrated operations against banks, governments and companies. For instance, in 2011, Anonymous group broke into HBGary web server [20]. The attackers collected a large number of password hashes, which belong to the company's CEO and COO. Moreover, a noticeable security company,

named Spamhaus [1], was subjected to a distributed denial-of-service (DDoS) attack in 2013, resulting in 300 Gbps stream of attack [67]. Furthermore, the largest DDoS attack ever launched in the world [2] is the recent 400 Gbps DDoS attack, reported in February 2014. In addition, coordinated attacks are generally made possible through botnets. The latter instrument compromised machines through Command-and-Control (C&C) servers. Botnets are generally used as a delivery platform of cyber-attacks. The latter are made possible by enjoining widespread bots to perpetrate malicious activities such as stealing sensitive information, reconnaissance of networks, taking advantages of existing vulnerabilities, DoS attacks, etc.

In this context, the detection and the filtering of malicious traffic, emanating from these compromised machines or their C&C servers, are essential for the early interception of these attacks as well as their mitigation. In this thesis, we address the problem of fingerprinting malicious IP traffic. By fingerprinting, we refer to: (1) the detection of malicious network flows and, (2) the attribution of the detected flows to malware families that generate them. As requirements, we formulate the following desideratum:

- Detect malicious IP traffic even in the presence of encryption;
- Achieve a high accuracy with low false alerts in both detection of malicious flows and attribution to malware families.

The detection of maliciousness spans over host-based detection and network-based detection. Regarding the host-based detection, despite the fact that such approach has been widely addressed by the security research community, it inherits the problem of disassembling malware binaries

[1] http://www.spamhaus.org/

[2] http://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack

in order to digest their functionality and structure [29, 37, 40, 68] since many tools are deployed to obfuscate and pack malware. Another host-based approach focuses on monitoring malware activities and obtaining malware behavioral profiles based on certain system calls or API calls. Generally, such solutions are OS dependency. Moreover, the host-based detection is a solution that is dedicated to each host within a given network. Thus, more resources are needed to handle it.

On the other hand, conventional network-based detection techniques fail to satisfy the previously stated requirements. Indeed, network-based detection techniques, whether they are signature-based or anomaly-based, generally resort to Deep Packet Inspection (DPI) [12] to extract payloads to compare them with malicious signatures. By doing so, they fail the scalability requirement. Moreover, if the traffic is encrypted, they fail to detect maliciousness. In addition, it is a well-known fact that anomaly-based detection tends to have a high level of false alerts.

1.2 Objectives

The objective of this research is to elaborate a framework for the detection of malicious communication with attribution to the malware family that is responsible for its generation. This framework should be applicable to both clear-text and encrypted traffic. The elaboration of this framework is predicated by the fact that compromised machines generate specific flows that can be distinguished from the benign ones. Accordingly, we aim at answering the following questions:

- How to detect maliciousness at the network traffic level?
- Which malware family is responsible for the generation of a detected malicious traffic?

Bringing forward answers to these questions would help security analysts to discover infected machines within their network or to detect the existence of a botnet communication that stems from or to a C&C server. Knowing the underlying malware family would help identifying the mitigation actions (e.g., quarantining a compromised machine, sinkholing, order C&C server take down).

1.3 Approach Overview

In this section, we describe the architecture of our fingerprinting framework. As shown in Figure 1, the framework consists of two main components: (1) Malicious traffic detection, and (2) Malware family attribution. In the following, we present an overview of these two components.

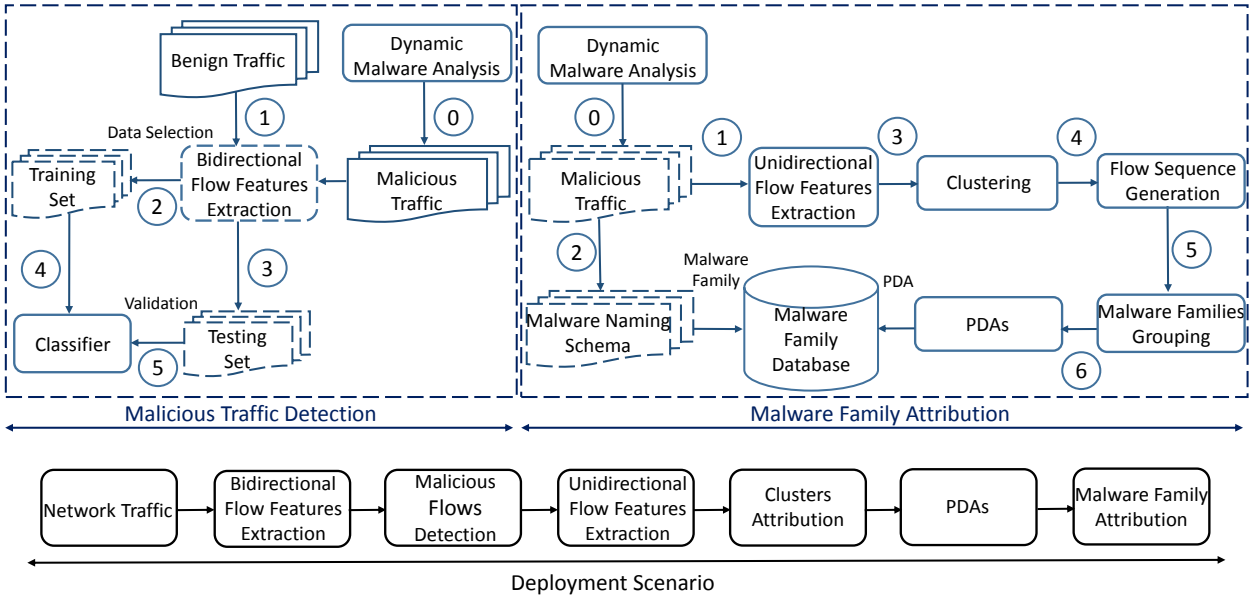


Figure 1: Framework Overview

First, for the detection, we extract bidirectional flow features from malicious traces, generated from dynamic malware analysis, together with benign traces, obtained from trust public repository such as Wisnet [11] and a private corporate network. These features are used by classification

algorithms to create models that segregate between malicious and benign traffic. Relying on the flow features that exploit only high-level properties of traffic flows and therefore does not rely on deep packet inspection makes our technique effective even in the presence of encrypted traffic.

Second, for the attribution, we create signatures for each malware family to segregate between them. The derivation of these signatures is achieved in three steps: First, we use clustering to group and label malicious flows based on their network behaviors. Second, we generate sequences of labeled flows for each malware family. Families that share the same sequence patterns are grouped together using sequence mining. Third, we generate PDAs out of sequences representing groups of malware families. Indeed, these PDAs act as generic signatures for malware family groups.

1.4 Contributions

The major contributions of this thesis can be summarized as follows:

- Elaboration of a technique for the detection of malicious flows at the network level. The proposed technique can be applied for both plain-text and encrypted traffic.
- Definition and derivation of pushdown automata as signatures that capture malicious communication.
- Elaboration of a technique that attributes a detected malicious flow to the families that are responsible for its generation.

- Elaboration of a technique that performs attribution even in the event of an incomplete malicious communication.

- Validation of the proposed detection and attribution techniques by investigating their accuracies as well as their ratios of false alerts. Besides, we explore the robustness of the proposed models through their application to various datasets.

1.5 Thesis Organization

The remainder of this thesis is structured as follows:

- Chapter 2 introduces the background literature and related work. We first describe the different methods and techniques that analyze malware specimens. Further, we detail the existing dynamic analysis tools since they are used in our work. Moreover, we summarize the existing methods for badness traffic detection and prevention.

- Chapter 3 details the adopted approach for malicious traffic detection. The latter consists of five main steps: (1) Dynamic malware analysis, (2) data labeling, (3) extraction of traffic flow features, (4) classifiers training, (5) evaluating the models produced by the classifiers. In addition, we explore the robustness of our approach by measuring the detection performance through applying the detection approach on different datasets.

- Chapter 4 explains our proposed mechanism for the attribution of the detected malicious traffic to the malware family that is responsible of its generation.

- Chapter 5 provides concluding remarks together with a discussion of future work.

Chapter 2

Literature Review

In this chapter, we review the literature related to the topics addressed in this thesis. We start by reviewing the different malware analysis techniques that are used to get more insights about malware specimens. Then, we present the existing malware analysis tools. Further, we detail the research contributions on maliciousness detection, at the network and the system levels, by providing the concept as well as the pros and cons of each approach. In addition, we explain the traffic classification mechanisms since our research tends to classify the malicious traffic from the benign one.

2.1 Malware Analysis Techniques

There are various techniques that are used to monitor the behaviour of malicious programs. In the following, we describe the different techniques, which fall into static and dynamic malware analysis techniques.

- **Static Malware Analysis:** Static malware analysis aims at dissecting, without execution, malware samples to find out the different functionalities that are hidden in malware binaries. Static analysis can be applied to the source representation of a program. Static analysis is used to harvest relevant information about malware. For instance, a call graph gives an analyst an insight about malware structure and which function may be invoked in the code. By analyzing binaries, an analyst can be confronted with binaries that implement obfuscation, packing and self-modification, which makes the static analysis even more challenging.

- **Dynamic Malware Analysis:** Dynamic malware analysis consists of analyzing the actions of a program while it is being run in a controlled environment. There are several techniques that are related to such analysis and they spin over:

    1. *Function Call Monitoring:* A function call relies on the analysis of a code that performs actions intended for different tasks. These functions tend to be re-usable in different versions of malware. One possible way to analyze malware behavior consists of intercepting functions. Such a method is called hooking. The latter allows to log function invocations and analyze the input/output parameters. There are many approaches for the implementation of the hooking function, depending on the availability of source code. If the latter is available, hooks can be inserted into appropriate places. Another technique is to use binary rewriting that generally falls into two techniques: rewriting monitored functions to call hook functions or modifying all call locations to invoke the hook. The hook function can access the original arguments on the stack and monitor them. Moreover, if a function is invoked through a DLL function pointer, the value of

a pointer can be changed to point to the hook function. Hunt and Brubacher [36] introduced the Detours library to apply function rewriting in order to implement the hooking function. Their technique consists of creating a trampoline function that contains overwritten instructions. This function contains an unconditional jump to the original one. Detours library has the ability, either to modify the binary before execution, or manipulate the in-memory images of loaded binaries.

2. *Function Call Traces:* This technique tends to monitor the inputs and outputs of function calls. The trace lies in the set of functions that were invoked by the program with passed arguments. These traces are used to create abstract representations of malware behavior. In [25], the authors represented call traces with graph representations, which allowed to compare the behavior of malicious programs with legitimate software. Thus, analysts can find out malicious instances of the same malware families within unknown samples. Xu et al. [69] used traces of known malware to detect polymorphic variants of unknown samples. The authors applied sequence alignment technique to compute similarities of function traces. Such a technique can take a considerable running time in order to calculate differences and similarities between traces.

2.2 Malware Analysis Tools

This section presents an overview of existing tools used to analyze potential malicious programs. We describe each tool in order to reflect valuable insights about them. The described tools fall mainly into two categories: static and dynamic analysis tools. In [59], the authors descripe the

static analysis tools that can be used to get insight about the malware specimen. Table 1 illustrates the list of the tools. Further, we descripe the dynamic malware analysis tools.

Table 1: Static Malware Analysis Tools

Tool	Description
Strings	Retrieve strings from the malware sample
PEiD	Detecting packers files
Upx	Unpack packers files
Dependency Walker	To explore the dynamically linked Function

- **WILDCAT:** In [63–66], Vasudevan and Yerraballi introduced a set of research efforts that depict WILDCAT framework. The latter consists of many components that implement binary instrumentation, stealth breakpoints and localized execution. This framework is designated to conduct coarse, fine-grained malware analysis.

- **Ether:** In [28], the authors proposed a malware-analysis framework based on hardware virtualization. It is stated that this framework is transparent since it is integrated in Xen hypervisor residing in a higher privilege level in comparison with normal operating systems. Ether monitors instructions, memory writes and system calls. It implements a mechanism for Windows XP to analyze a specific process. The monitoring of instructions is turned on by setting the trap flag of the CPU. As a result, Ether is able to trace executed instructions by the guest operating system.

- **Hookfinder:** Malware usually implements hooking techniques to evade detection. This is one of the manners to infect a system in a stealthy way. These techniques are usually done by spyware and rootkits. Hookfinder is a tool, which is able to detect hooking techniques

and report them. This tool employs data and address taint-tracking in a modified version of Qemu [16] emulator.

- **Joebox:** Joebox [21] is a sandbox technology, which creates logs that enclose information related to different actions done by malware. These logs provide information about file system changes, registry and system activities. Joebox does not rely on virtual machines for emulation techniques. It runs specifically on physical machines. This technology is based on server-client model, where the server coordinates with malware analysis clients. The output of the analysis is collected by the server. Joebox hooks system calls and user-mode API.

- **Norman Sandbox:** Norman Sandbox [58] is a solution that runs malware samples in a virtual environment that simulates Windows operating systems. The environment consists of virtual hosts connected to the Internet. The virtual hosts are tweaked to support connectivity as well as APIs that are used commonly by malware. Since the malware executes in a dynamic virtual environment, it cannot hide itself by using packing techniques.

- **Multiple-Path Exploration:** Automatic dynamic malware analysis tools generate their reports based on a single execution trace of the sample under analysis. The use of logic bombs allows malware to only reveal its malicious behavior based on arbitrary constraints. For example, a malware sample could postpone its malicious activities until a certain date is reached or stop executing if necessary files cannot be found on the infected system. To overcome this shortcoming, Moser et al present a tool that is capable of exploring multiple execution paths for Windows binaries. This tool recognizes a branching point whenever a control-flow decision is based on data that originates outside the monitored process. This

data can only be introduced to the process via system calls. Thus, a branching point is detected if a control-flow decision is based on a return value of a system call (e.g., the current system time). Every time such a situation occurs, the tool takes a snapshot of the running process that allows the system to reset to this state. Execution is continued and after a time-out (or process termination), the system is reset to the recorded snapshot. Then, the value that is responsible for the control-flow decision is manipulated such that the control flow decision is inverted, resulting in the execution of the alternate path.

- **ThreatAnalyzer Sandbox [4]:** is a sandbox technology, which creates reports that enclose information related to different actions done by malware. These reports provide information about file system changes, registry and system activities in addition to the network activities. ThreatAnalyzer can be used on real or virtual machines. This technology is based on server-client model, where the server coordinates with malware analysis clients. The output of the analysis is collected by the server. In my research thesis I adopt the ThreatAnalyzer as sandbox for two main reasons : (1) It provides all the information I need for my research, (2) Supporting the virtual and real machines as sandbox clients. More details about how and why I use this sandbox are discussed on chapter 3.

2.3 Network-based Detection

This section presents existing techniques on maliciousness detection at the network level. These techniques can be classified into the following:

- **Signature-based Detection:** It aims to analyze packet payloads by looking for malicious content based on predefined signatures. In [42, 48, 60], the authors proposed approaches that reflect this technique. The latter has shown good results in terms of malware detection at the network level but it fails in capturing badness when the traffic is encrypted. Moreover, it needs sampling to preserve scalable detection in the presence of a large traffic. This technique is used in network intrusion detection systems (NIDSs) by monitoring packets and comparing them against attributes (signatures) stored in a database. The signatures are represented by a packet header rule and packet content rule. The packet header rule consists of source and destination IPs and ports. The packet content rule is a string or regular expressions pattern. NIDS such as Snort [7] and Bro [8] use regular expressions to define detection rules. It is noteworthy that regular expressions are more efficient and flexible in NIDS, in comparison with exact matching strings. Generally, signature based approach suffers from two main limitations: (1) the encryption of packets payloads makes signatures completely useless in the identification of harmful traffic, (2) fails in detecting zero day attacks.

- **Anomaly-based Detection:** This approach is mainly based on producing models that detect abnormal changes in a network based on the *normal use*. The first technique of anomaly detection in networks consists of using *statistical techniques*. In [61], the authors proposed a statistical anomaly-based detection system, namely, Haystack. It is considered as a prototype IDS. The authors created profiles for each feature where it is considered as normal. The features were modeled as Gaussian random variables. This tool runs offline. In Statistical Packet Anomaly Detection Engine (SPADE) project the authors used a frequency based approach to compute an anomaly score for each packet. If an anomaly score is higher

than a threshold, the packet is passed through a correlation engine, which detects scanning events on ports. Spade can generate high false positives since it considers unseen packets as attacks. Another statistical technique used to detect anomalies is *Principal Component Analysis (PCA)*. This technique deals usually with multi-dimensional and large datasets. The second technique, used for anomaly detection in networks, is the *machine learning approach*. Valdes et al. used Naïve Bayesian networks to detect intrusions on traffic bursts. Generally, the anomaly-based detection Suffers from a high level of false alerts.

- **Mining-based Detection:** This approach uses machine learning algorithms, including supervise and unsupervised machine learning, in order to infer the badness. Unlike the anomaly-based approach, the mining-based one does not rely only on the benign traffic. The main intent of existing work that introduced such technique is the identification and detection of botnets. Some approaches that detect botnets rely mainly on heuristics observed on botnet architecture like in [18, 26, 44] for IRC botnet and [22, 38, 49] for P2P botnets. In [39], Karasaridis et al. put forward an approach to identify botnet C&Cs by combining heuristics characterizing IRC flows, scanning activities and botnet communication.

  In [33], the authors introduced BotHunter, which models all bot attacks as a vector enclosing scanning activities, infection exploit, binary download and execution, C&Cs communication. The tool is coupled with Snort IDS with malware extensions to raise alerts when different bot activities are detected. Other works used aggregation to detect botnets. For instance, BotSniffer [34] tried to show that infected hosts have spatial-temporal similarity. It pinpoints to suspicious hosts that have malicious activities such as sending emails, scanning

and shared communication payloads by using shared bi-grams in IRC and HTTP botnets. In [32], the authors put forward BotMiner, which aim to identify and cluster hosts that share common characteristics. In [62], the authors introduced a novel system, namely, BotFinder, which detects infected hosts in a network by considering high-level properties of the botnet network traffic. It uses machine learning to identify key features of C&C communication based on bots traffic produced in a controlled environment.

The aforementioned works have shown interesting results and were considered as a source of inspiration for our work. Hence, we wanted to detect badness and attribute it to malware families. To do so, we considered the traffic generated from the dynamic malware analysis as a maliciousness ground truth instead of focusing on P2P, IRC or HTTP bots. Moreover, we considered high level properties of flows instead of payload of packets to model badness detection and its attribution. This is important since malicious traffic may contain encrypted payloads, which mislead badness detection and attribution.

2.4 System-based Detection

Similar to network-based detection, there are three main approaches for system-based detection, namely: *signature*, *anomaly* and *mining*-based detection. In what follows, we focus on the works that are based on dynamic malware analysis reports since they are close to our approach.

Rieck et al. [41] introduced an automatic analysis of malware based on machine learning techniques. They proposed two approaches, namely, clustering and classification of malware behavior collected from dynamic malware analysis. The first approach aims to discover similar classes of

malware with similar behavior, whereas the second technique tries to assign unknown malware to known classes. In their article, they put forth reason for the discovery of new malware classes as well as the discernment between known classes. As a result, they proposed a framework for the automatic analysis of malware using clustering and classification. They claimed that their clustering and classification are scalable. They also made an incremental analysis of malware behavior by joining clustering and classification. Moreover, they assessed their framework with real malware and stated that they perform effectively.

This plethora of works show the benefits of using dynamic malware analysis technology to monitor and characterize malware behavior. In the prevailing of this thought, we have decided to use such capability to fingerprint malicious traffic based on dynamic malware analysis.

2.5 Traffic Classification

There are three traffic classification approaches, namely *port-based detection* of application layer protocols, *Deep Packet Inspection (DPI)* techniques and *data mining approach*.

- **Port Based Detection:** Network tools such as Wireshark [10] tends to detect protocols based on ports that are assigned conventionally to applications listed in IANA [6]. The concept of ports being assigned to application protocols is no more valid. Consequently, the concept of Deep Packet Inspection (DPI) has been introduced.

- **Deep Packet Inspection:** DPI is a technique that digs into TCP/IP payload to identify types of packets. It searches for patterns to infer the service or application being requested. Many

works have proposed to identify network traffic based on DPI. The foundation of DPI systems is based on string matching algorithms. However, these algorithms affect the matching performance of identification systems. For this purpose, other works suggested that packet payload signatures can be used to generate identification rules. This approach was mainly used to identify non-conventional ports applications like P2P programs. L7-filter [1] is a tool used for the identification of the application layer protocols through content signatures or connection patterns. The L7-filter is a signature matching system able to identify 120 types of application protocols. However, it works as a plugin of the Linux iptables system. nDPI [12], known previously as OpenDPI, is an open source and cross-platform library designed to monitor application protocols. The main intent of this library is to detect known protocols on non-standard ports. nDPI detects around 140 protocols and it has the ability to load an encryption decoder to identify SSL-based protocols such as Citrix and Apple iCloud.

- **Data Mining Approach:** The data mining approach falls into clustering, classification and hybrid techniques. The clustering stands for works that rely on unsupervised learning techniques. The classification centers on using supervised machine learning algorithms. The hybrid technique combines both supervised and unsupervised learning techniques.

    1. *Clustering Approach:* in [45], McGregor et al. published a work that applied IP traffic classification using the EM algorithm [27]. The approach consists of clustering traffic with similar properties into different application types. In [71], Zander et al. used an unsupervised Bayesian classifier, namely, AutoClass [24] to determine clusters from training data. The features are derived from bi-directional flows by using NetMate [70].

In [17], Bernaille et al. proposed using Simple K-Means, Gaussian Mixture Model and Hidden Markov Model to classify different types of TCP-based applications. The authors used the first few packets of the traffic flow in order to do early detection of traffic flow. Erman et al. [30] dealt with traffic classification at the network level. They classify flows by using unidirectional flow information. Their approach proposed the use of K-Means clustering algorithm.

2. *Supervised Learning Approach:* in [56], Roughan et al. used nearest neighbors, linear discriminate analysis and Quadratic Discriminant Analysis algorithms to link network applications to QoS traffic classes. The classification used 10-times cross validation to evaluate models. In [47], Moore and Zuev applied Naïve Bayes technique to classify Internet traffic based on application protocols. They enumerated 249 flow based features listed in [46]. The authors used two methods, namely Naïve Bayes Kernel Estimation and Fast Correlation-Based Filter to reduce the feature space and improved the classifier accuracy. The authors extended their works in [15], where they use Bayesian neural network approach. The paper introduced a list of features and ranked them according to their importance. In [14] the author classify the skype encrypted traffic using packets and flow features through classification approach. The author concludes that the machine learning C5.0 can be a good alternative for traffic classification.

3. *Hybrid Approach:* Erman et al. [30] put forward a semi-supervised traffic classification approach which aggregates unsupervised and supervised methods. The author proposed a classification method, which consists of two steps. The training step where labeled flows and unlabeled flows are input to k-means cluster algorithm. The second

step consists of mapping the clusters to different known classes. In order to map a cluster to a class, the authors used a probabilistic assignment, in the name of the maximum likelihood estimate. The intent behind exploring different works related to fingerprinting application protocols, is to understand the different techniques that have been used on different works and how application protocol fingerprinting supports our work in building protocol sequence patterns for malicious traffic. Such work showed that machine learning approach achieved promising results with data collected from network traffic traces. As such, we decided to use such artifact to conduct a thorough analysis on traces collected from malware dynamic analysis.

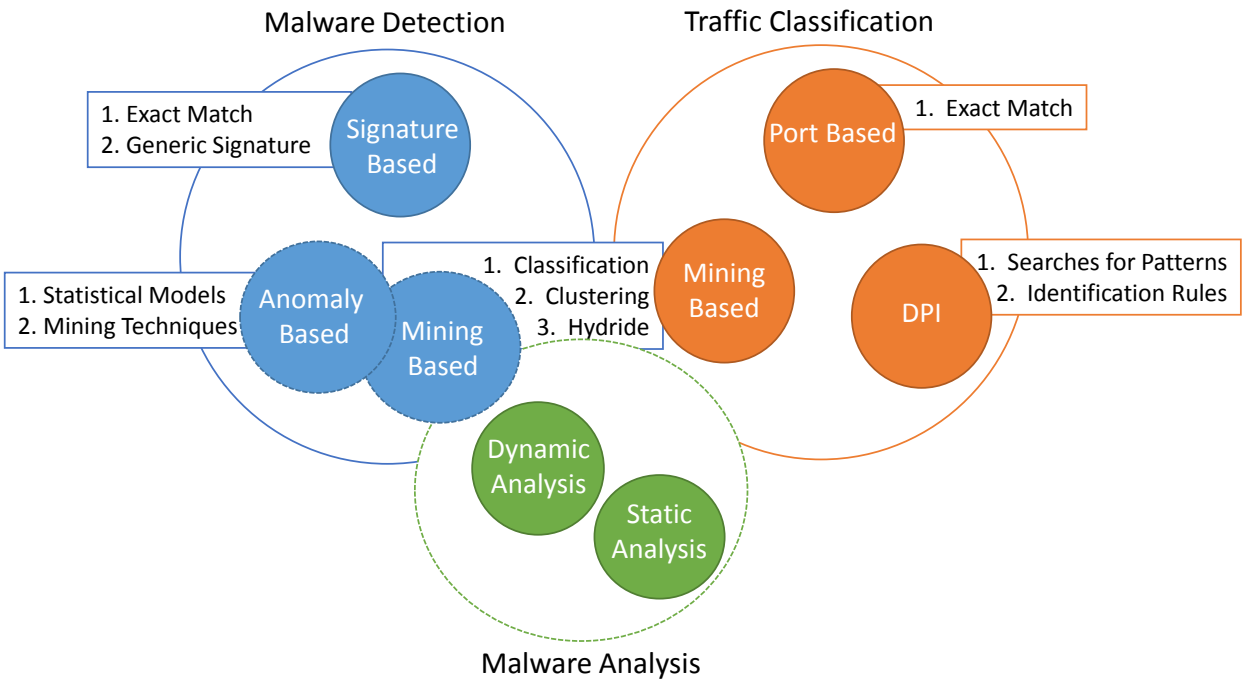


Figure 2: Related Work Overview

Figure 2 summarizes the related work. To conclude , we review in this chapter the two main techniques for malware analysis: static and dynamic. Further, we detailed the different tools under

each category. In our research we use dynamic analysis technique. Moreover, we describe the three main approaches for malware detection at the network and system levels. In addition we explain the traffic classification methods. We did inspire from the existing works to accomplish our thesis objectives and we did learn the following lessons that we take it on account while desining our proposed approach:

1. The list of flow features that have been shown to produce distinguishing characteristics when applied on network data.

2. In order to classify the encrypted traffic, it is better to use packets header information.

3. To have a scalable solution, we should not go deep into the payload level.

4. Data dependency of the mining solution.

5. The limitation of the signature and anomaly based detection.

In the following chapters we introduce our proposed method for malicious traffic detection and attribution.

Chapter 3

Malicious Traffic Detection

3.1 Introduction

Badness detection at the network level is meant to isolate communication sessions that correspond to malicious activities. These sessions embed flows that are used to perpetrate various malicious activities (e.g., DDoS, credential theft). Generally, these flows are intermingled with a large portion of IP traffic that corresponds to benign activities over computer networks. As such, badness detection amounts to a segregation between malicious and benign flows. In this respect, we represent flows through a set of attributes (features) that capture their network behaviors. By leveraging these features, we create classifiers that automatically generate models to detect malicious traffic. In the prevailing of this thought, we define three phases to infer badness at the network level: (1) Selecting and extracting the bidirectional flow features and labeling of the traffic (malicious and benign), (2) training via machine learning algorithms, and (3) evaluating the classifiers produced

by these algorithms. These steps together with the exploration of the generalization of the proposed method are addressed in details in this chapter. Figure 3 illustrates the proposed method for malicious traffic detection.

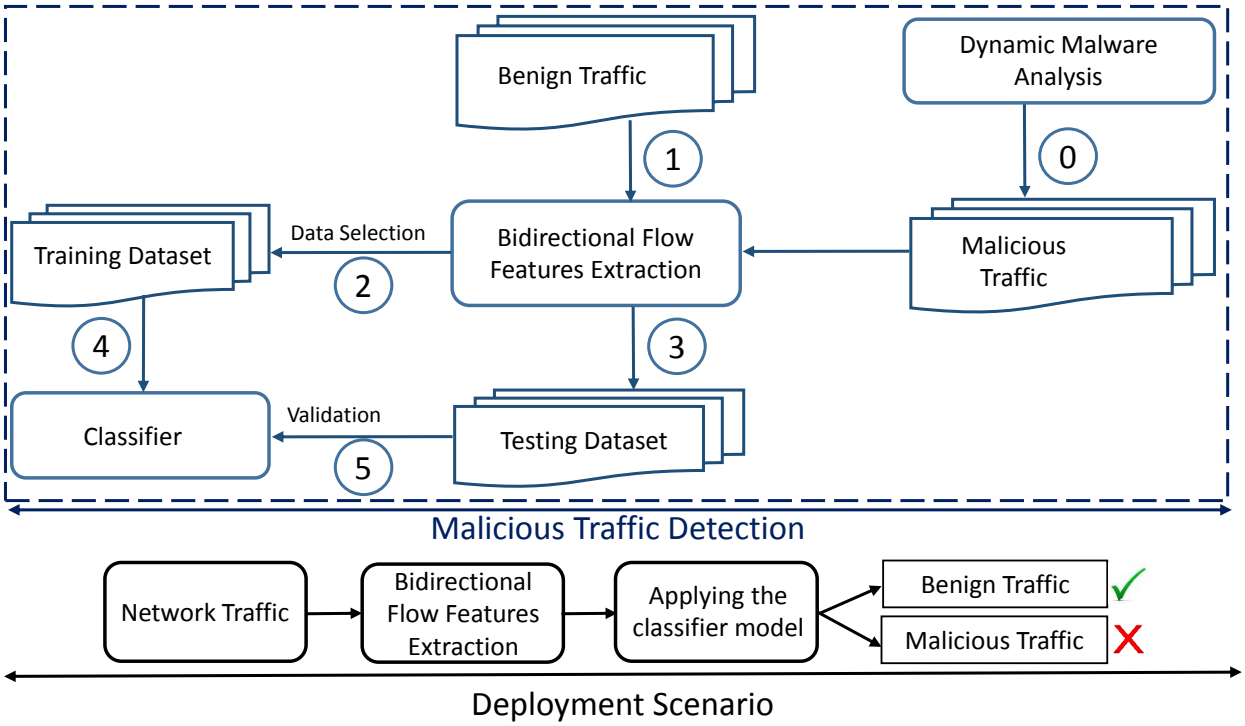


Figure 3: Malicious Traffic Detection Framework Overview

3.2 Malicious Ground Through

The aim of this thesis is to establish a technique to infer badness at the network level. To achieve the previously mentioned objectives, we need two types of network traces: malicious and benign traffic. To generate malicious traffic, we dynamically analyze a large (2 million+) collection of malware binaries in a controlled environment, aka sandbox. The latter is used as a ground truth for maliciousness. Moreover, we gather benign traffic from several sources, including network traces

from public repositories and traffic from a corporate network [11]. In the following, we explain how we generate and/or collect these network traces.

3.2.1 Malware Feeds

We receive malware samples on a daily basis, on a daily basis, an average of $4,560$ malware samples from our partner ThreatTrack Security [4]. So far, we accumulated a collection of more than 4.6 million malware binaries since 2011. For this work, we experimented with 2.46 million samples collected between Jan 2011 and June 2012 when we started this research. Figures 4 illustrates the malware counts recorded during the period in question.

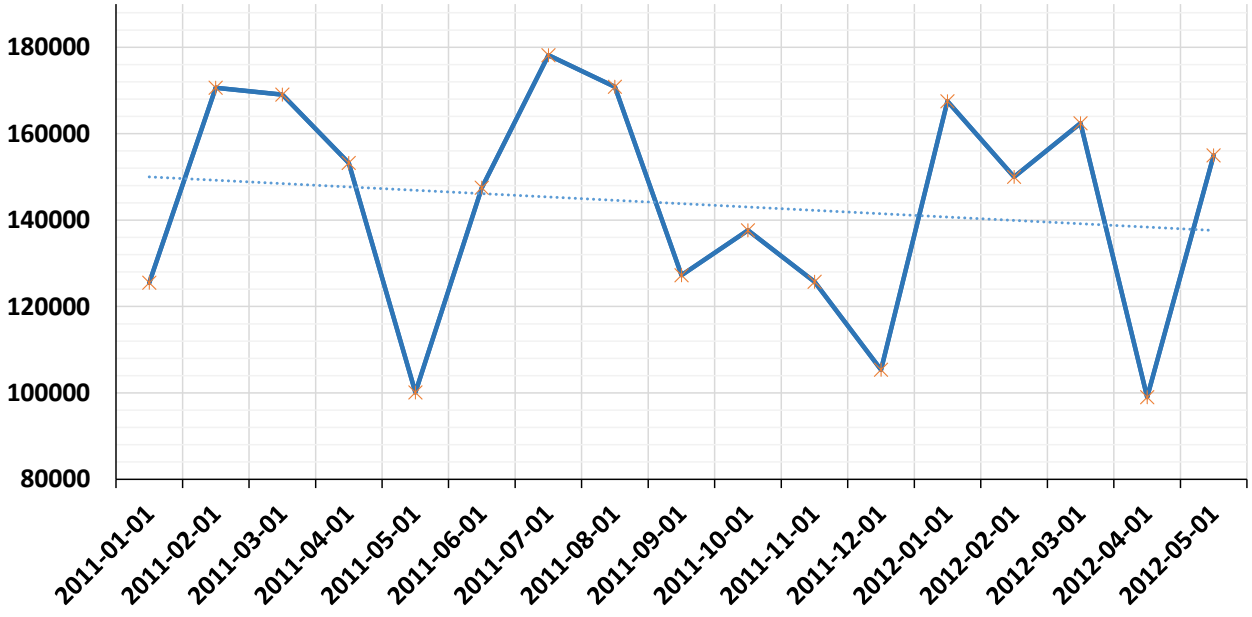


Figure 4: Malware Feeds

3.2.2 Malware Analysis Topology

We have executed the collected malware samples in a controlled environment (sandbox) to generate representative malicious traffic. The latter is used as a ground truth for maliciousness fingerprinting. The sandbox is based on a client-server architecture, where the server sends malware to clients. The sandbox clients are responsible of executing malware during three minutes. We chose this period to be able to handle the daily received malware feeds. The clients monitor the behavior of each malware and record it into report files. These files contain malware activities such as file activities, hooking activities, network activities, process activities and registry activities. It is important to mention that the dynamic analysis setup allows malware to connect to the Internet to generate inbound/outbound malicious traffic.

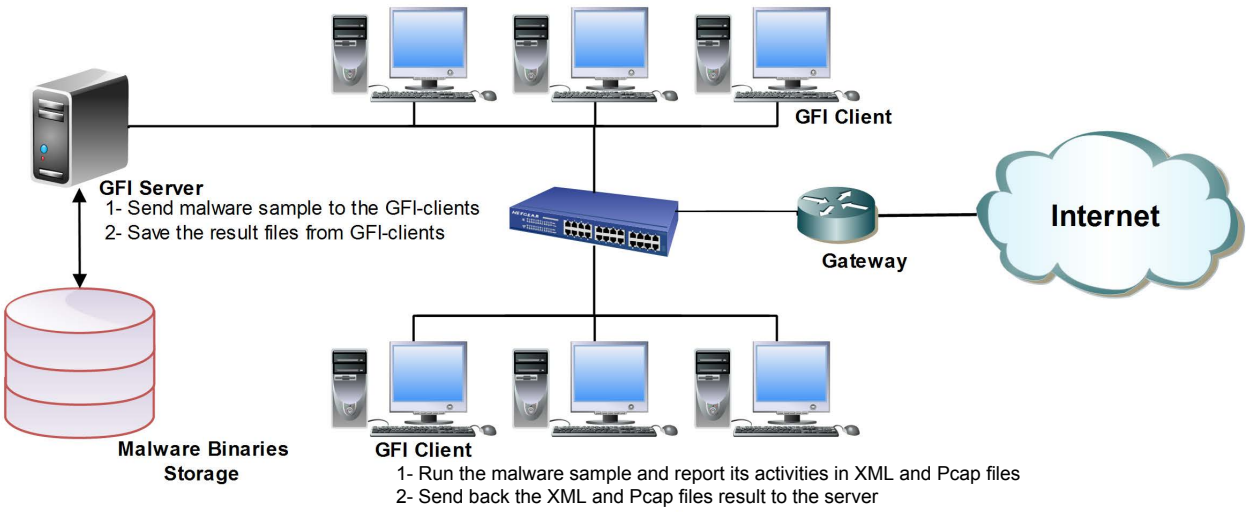


Figure 5: Dynamic Malware Analysis Topology

The dynamic malware setup is a network, which is composed of a server and 30 client machines. The server runs with an Intel(R) Core^{TM} i7 920@2.67 GHz, Ubuntu 11 64 bit operating

system and 12.00 GB of physical memory (RAM). Each client runs with an Intel(R) Core^{TM} 2 6600@2.40 GHz, Microsoft Windows XP professional 32 bit operating system and 1.00 GB of physical memory.The reason underlying the use of physical machines, some malware samples detect that they are being run in a virtual environment and therefore do not exhibit their malicious behaviors. Figure 5 illustrates the dynamic malware analysis topology. As a downstream outcome of the aforementioned dynamic analysis, we have gathered the underlying traffic Pcap files that were generated. The dynamic malware analysis has generated around $100,000$ Pcap files, labeled with hashes of malware, which correspond to a size of 3.6 GO.

Moreover, in our work, we relied on Kaspersky Naming Schema [1] to map between malware samples and the corresponding malware names and families. The reasons behind choosing Kaspersky naming schema are as follows: (1) We have noticed that this naming schema managed to cover the naming of the majority of malware samples. (2) The malware naming schema provided by Kaspersky follows exactly the malware convention name (Type.Platform.Family.Variant). Figure 6 shows an example of Kaspersky malware name.

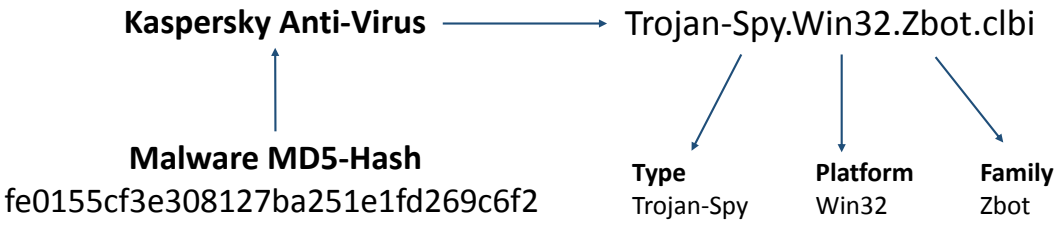


Figure 6: Example of Kaspersky Malware Name

We have identified 305 malware families. Each family is mapped to a set of malicious network traces indexed per malware hash. Figure 7 and Figure 8 illustrate the frequent top 10 malware families and malware variants respectively.

[1] http://usa.kaspersky.com/internet-security-center/threats/malware-classifications

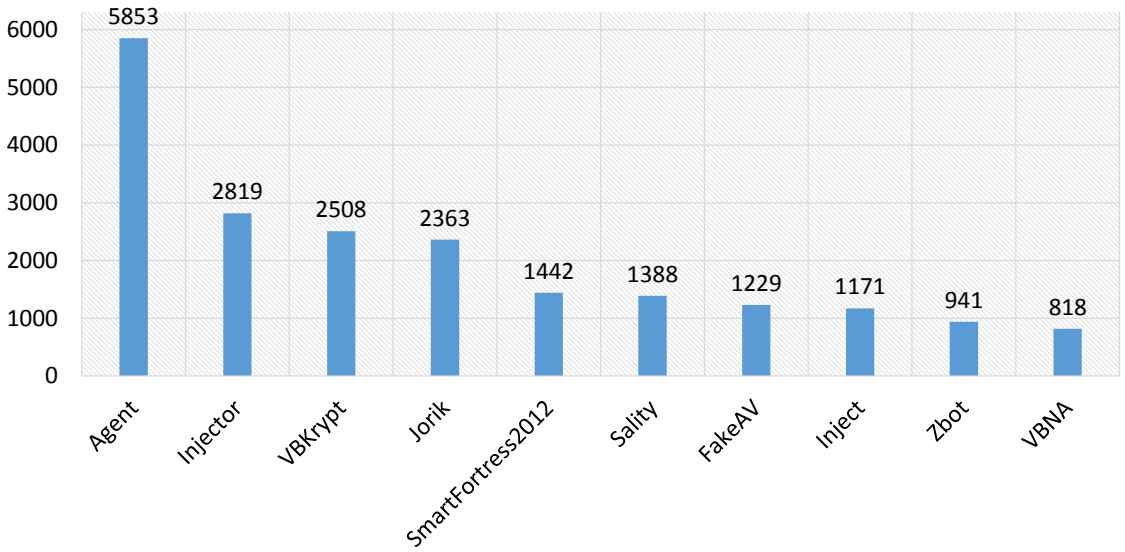


Figure 7: Frequent Top 10 Malware Families

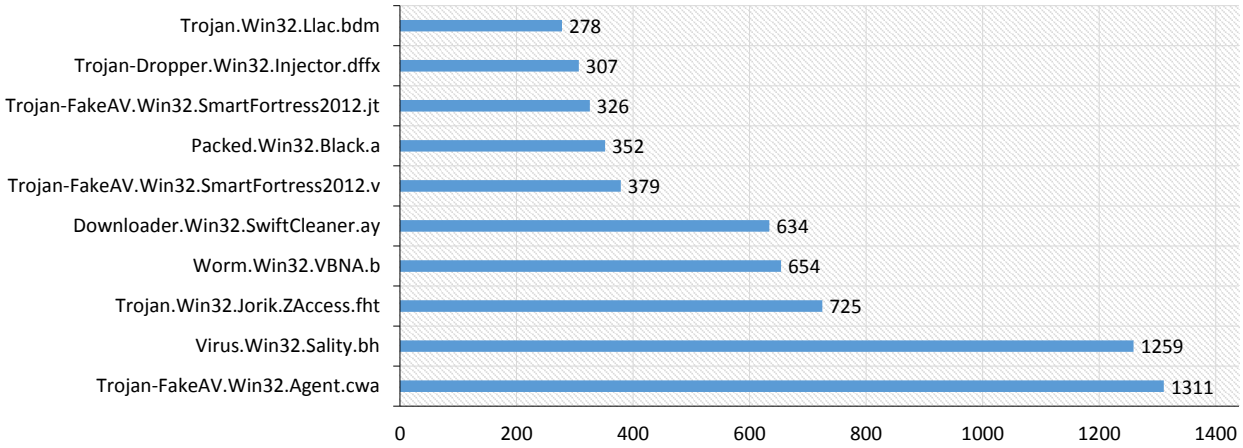


Figure 8: Frequent Top 10 Malware Variants

3.3 Network Traces

For the purpose of building a classification model that distinguishes between malicious and benign traffic, we collect benign traffic from WISNET [11] and a private corporate partner. These datasets

have been built to evaluate Intrusion Detection Systems in terms of false alerts and detect anomalies in network traffic. In our work, we use such datasets to build a baseline knowledge for benign traffic. These datasets have been used together with the malicious traffic dataset to assess classification algorithms in terms of accuracy, false positives and negatives. Table 2 shows the number of benign flows in each dataset. In addition to the benign traces, we have also used the malicious traffic represented in Chapter 3.2.

Table 2: Datasets Description

Source	# Bidirectional Flows	Description
WisNet (Home)	$10,513 flows (85 Mo)$	Traffic collected from a residential setting
WisNet (ISP)	$65,000 flows (1.1 Go)$	Traffic collected from a research laboratory
WisNet (SOHO)	$16,504 flows (1.3 Go)$	Traffic collected from the edge router of an ISP
Private	$64,004 flows (5.6 GO)$	Traffic collected from a private company
Malicious Traffic	$96,004 flows (3.6 GO)$	Traffic generated from the Dynamic Malware Analysis

3.4 Bidirectional Flow Feature Extraction and Traffic Labeling

We capture malicious network traces from the execution of malware binaries in ThreatTrack sandbox. Accordingly, we label these traces as malicious, while the clean traffic traces, obtained from trusted third parties [11], are labeled as benign. Flow features are extracted from these labeled network traces to capture the characteristics of malicious and benign traffic. It is important to mention that these features can be extracted even when the traffic is encrypted since they are derived from flow packets' headers. The exploited flow features are mainly based on flow duration, direction, inter-arrival time, number of exchanged packets, and packets size.

Table 3: Description of Bidirectional Flow Features

	Features
1	Flow Duration
2	Number of forward packets
3	Number of backward packets
4	Protocol
5	Minimum inter-arrival time for forward packets
6	Maximum inter-arrival time for forward packets
7	Mean inter-arrival time for forward packets
8	Standard deviation inter-arrival time for forward packets
9	Total forward packets size
10	Minimum forward packets size
11	Maximum forward packets size
12	Mean forward packets size
13	Standard deviation forward packets size
14	Minimum inter-arrival time for backward packets
15	Maximum inter-arrival time for backward packets
16	Mean inter-arrival time for backward packets
17	Standard deviation inter-arrival time for backward packets
18	Total backward packets size
19	Minimum backward packets size
20	Maximum backward packets size
21	Mean backward packets size
22	Standard deviation backward packets size

A bidirectional flow is a sequence of IP packets that share the 5-TCP-tuple (i.e., source IP, destination IP, source port number, destination port number, protocol). The forward direction represents the outbound traffic, while the backward direction represents the inbound traffic. In terms of design and implementation, the module in charge of network traces parsing, labeling and feature extraction reads network streams using `jnetpcap API` [3], which decodes captured network flows in real-time or offline. This module produces values for different features from network flows. The resulted values are stored in features files that are provided to Weka [9], which is a data mining framework. The network traces parser represents each flow by a vector of 22 flow

features. Table 3 illustrates the description of the bidirectional flow features.

3.5 Traffic Classification

The feature files resulting from the previous phase are provided as input to classification algorithms. The intent is to build models that have the ability to distinguish between malicious and benign flows. To this end, we use machine learning algorithms, namely, Boosted J48, J48, Naïve Bayesian, Boosted Naïve Bayesian, and SVMs. The classification module is based on a Java wrapper that runs these machine learning algorithms. The module has two execution phases: learning and testing. In the learning phase, we build the classifier using 70% of malicious and benign traces. In the testing phase, we evaluate the classifier with the rest of the data (30%). It is important to mention that training and testing datasets do not overlap with each other. In the sequel, we give a brief overview of the classification algorithms. More details about these algorithms are provided in the Appendix.

Algorithm J48: It is a Java implementation of C4.5 classification algorithm [54]. The J48 algorithm [2] builds the tree by dividing the training data space into local regions in recursive splits. The split is pure if all observations in one branch belong to the same class. To split the training dataset, J48 computes the goodness of each attribute (feature) to be the node root. It begins by calculating the *information need* factor, which is the number of bits needed to represent the different classes. The expected *information need* is given in Equation (1).

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2 P_i \tag{1}$$

m: the number of classes in the dataset.

P_i: the probability that an arbitrary tuple belongs to a class C_i where $1 < i <= m$.

Next, the J48 computes the bits needed to represent the classes in each sub-dataset after the split by each attribute. That information is given by the *information need* per attribute factor. The smallest need will be chosen, therefore, the corresponding attribute will be used to split the training dataset. The *information need* per attribute is given in Equation (2).

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j) \tag{2}$$

A: the representation of a given attribute, which has v different values.

D: the dataset. $|D|$: the number of instances in D.

$|D_j|$: the number of instances after splitting D per attribute A by using a value j.

If the split is not pure, the same process will be used to split the sub-dataset into pure classes. The split stops if each sub-dataset belongs to one class (pure split) or the entire attributes are already used. The decision tree result is composed of nodes (the attributes) and terminal leaves (the classes). That will be used to identify the unseen data when the model is deployed [35].

Naïve Bayesian Algorithm: It is based on Bayes' theorem [35]. It is a statistical classifier, which outputs a set of probabilities that show how likely a given tuple (observation) may belong to a specific class [35]. Naïve Bayesian starts by computing the probability of an observation that belongs to a specific class by using Equation (3). Naïve Bayesian assumes that the attributes are mutually independent.

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \tag{3}$$

X: the observation.

C_i: the class i.

$P(X|C_i)$: the probability that an observation X belongs to the class C_i.

n: the number of attributes (features).

x_k: the element k in the observation X.

$P(x_k|C_i)$: the probability that an element x_k belongs to the class C_i.

Once the probabilities are computed, Naïve Bayesian associates each observation with the class that has the higher probability with it. Naïve Bayesian is an incremental classifier, which means that each training sample will increase or decrease the probability that a hypothesis is correct. This option makes updating the Naïve Bayesian model very easy with new observations. In addition, Naïve Bayesian exhibited speed in comparison with other machine learning algorithms when it is applied to large datasets.

Boosting Algorithm: It is a method that is used to construct a strong classifier from a weak learning algorithm. Given a training dataset, boosting algorithm incrementally builds the strong classifier from multi-instances of a weak learning machine algorithm [31]. Figure 10 illustrates the different steps of boosting technique.

Boosting algorithm takes, as input, the training dataset and the weak learning algorithm. It divides the training dataset into many sub-datasets $(x_1, y_1), ..., (x_i, y_j)$, where x_i belongs to X (set of observations) and y_j belongs to Y (set of class attribute values), and calls the weak learning algorithm (WLA) to build the model for each sub-dataset. The resulted models are called decision stumps. The latter examine the features and return the decision tree with two leaves either $+1$ if the observation is in a class, or -1 if it is not the case. The leaves are used for binary classification (in

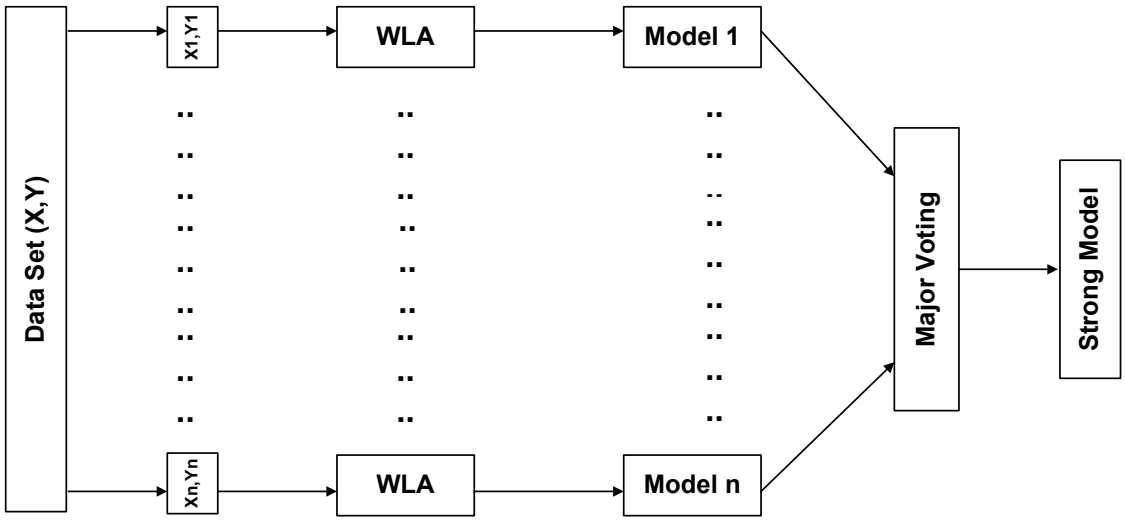


Figure 9: Boosting Technique

case the problem is multi-classes, the leaves are classes). Boosting algorithm constructs a stronger model by overlapping performance of the decision stumps by using a majority voting scheme between the decision stumps.

SVM Algorithm: The Support Vector Machines (SVM) algorithm is designed for discrimination, which is meant for prediction and classification of qualitative variables (features) [50]. The basic idea is to represent the data in a landmark, where the different axes are represented by the features. The SVM algorithm searches for the linear optimal separating hyper plane with the maximum margin within the hyper plane. In a feature space H, the solution is defined by Equation (4) representing the optimal problem resolution [50].

$$min_w(1/2)||w|| \tag{4}$$

such that: $\forall i > 0 : y_i < w, x_i > b \geq 1$, where w is the orthogonal vector.

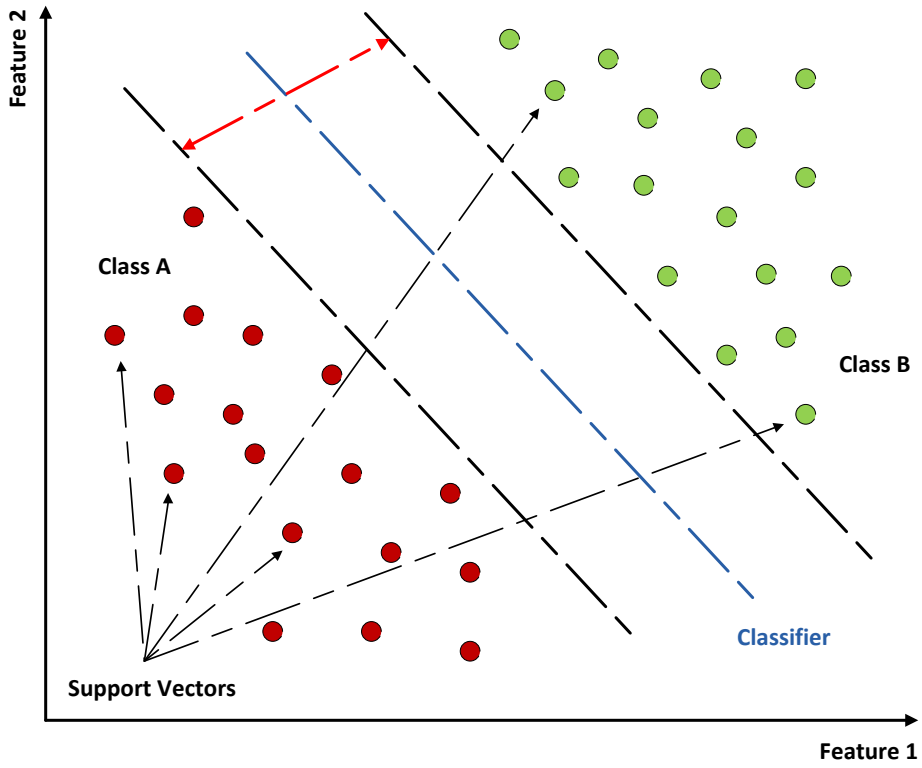


Figure 10: Support Vector Machines

3.6 Evaluation of the Classifiers

In order to measure the performance of machine learning algorithms, we use three evaluation benchmarks: Accuracy (AC), the False Positive Rate (FPR), and the False Negative Rate (FNR). The accuracy represents the total number of flows that are correctly classified among the total flows number. The accuracy is defined by Equation (5) as follows:

$$AC = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

On the other hand, the FPR and FNR reflect the total number of misclassified flows. Moreover, the FPR is dedicated for the out-class flows that are misclassified as in-class, and the FNR is

reserved for the in-class flow that are misclassified as out-class. Equation (6) formally defines FPR and FNR respectively. The aim is to achieve a high accuracy with low false (positive and negative) rates.

$$FPR = \frac{FP}{FP + TN} \qquad \text{and} \qquad FNR = \frac{FN}{FN + TP} \tag{6}$$

In order to address the generalization of our proposed method, we make a further step by applying the approach to different benign datasets with the same malicious dataset. Looking into the performance of machine learning algorithms under different scenarios helps us to measure the robustness of our proposed approach. It is important to mention that the WisNet benign datasets are collected in the period of December 2010 - March 2011, while the private benign dataset is collected on May 2013. The results of the classification experiments are provided in Section 3.7.

3.7 Experimental Results

In this section We detail the empirical results through answering the following questions:

Can we segregate between benign and malicious traffic?

The main intent is to identify a classifier with high accuracy and low false positives and false negatives. The results illustrated in Figures 11, 12, 13, 14, 15 and 16 demonstrate that Boosted J48 and J48 have shown better results than other machine learning algorithms. They have respectively achieved 99% accuracy and less than 1% false positives and negatives. The SVM algorithm has achieved good results with an accuracy ranging between 89% and 95%. In contrast, Naïve Bayesian and Boosted Naïve Bayesian algorithms have not achieved good results. As such, we can claim that Boosted J48 algorithm is a good artifact to differentiate between malicious and benign traffic.

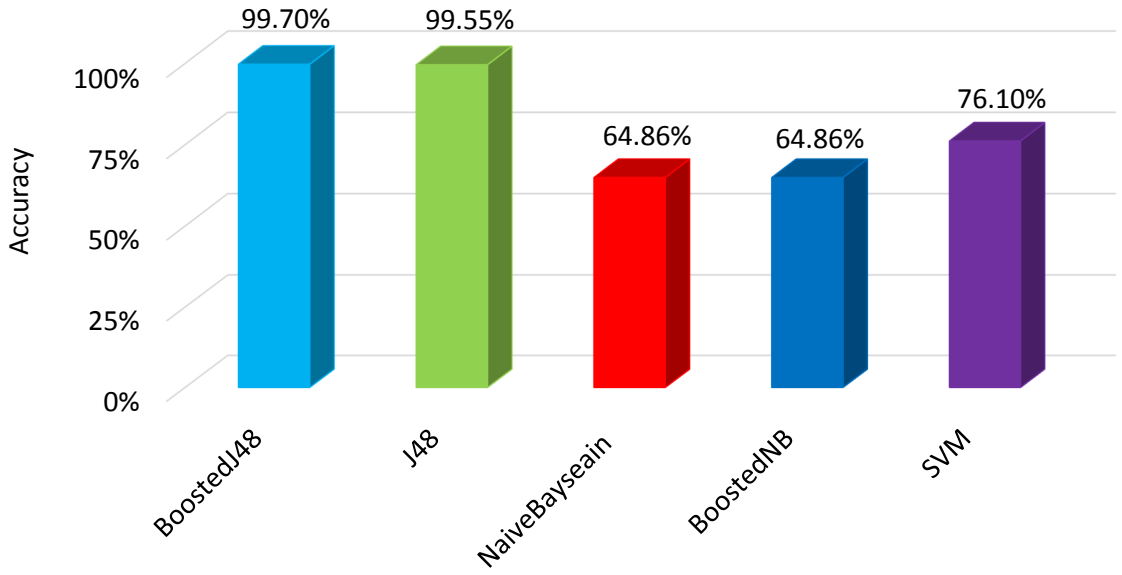


Figure 11: Accuracy Detection per Machine Learning Using Malicious and Benign (Home) Datasets

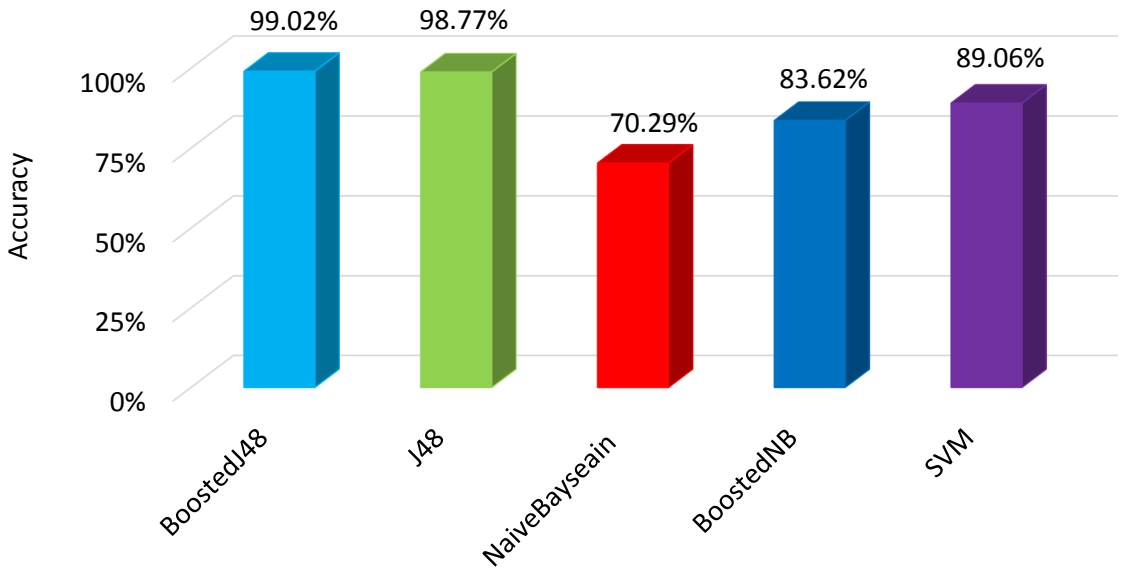


Figure 12: Accuracy Detection per Machine Learning Using Malicious and Benign (SOHO) Datasets

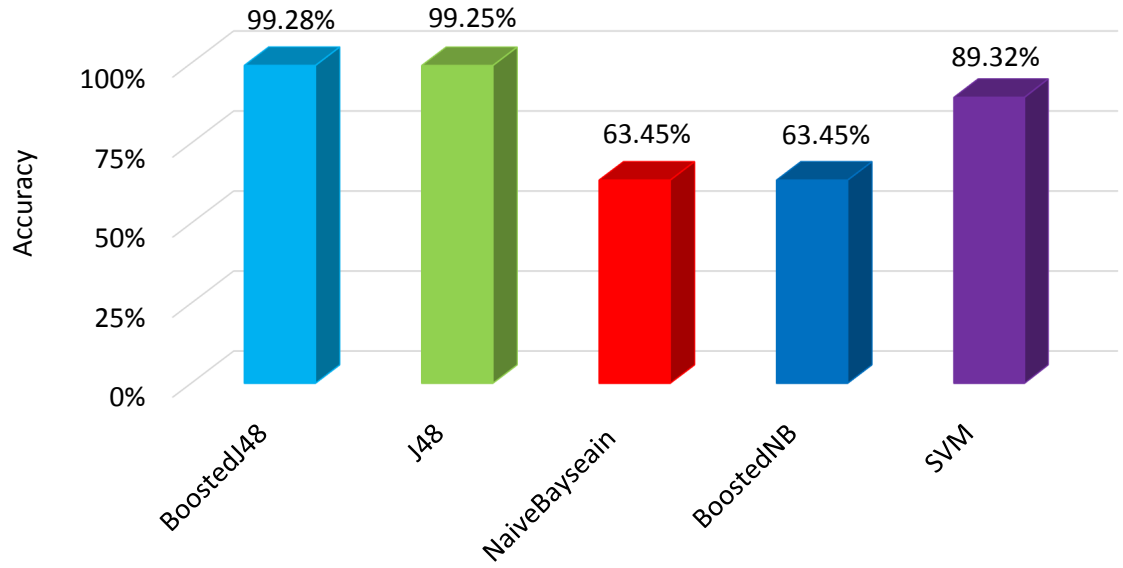


Figure 13: Accuracy Detection per Machine Learning Using Malicious and Benign (ISP) Datasets

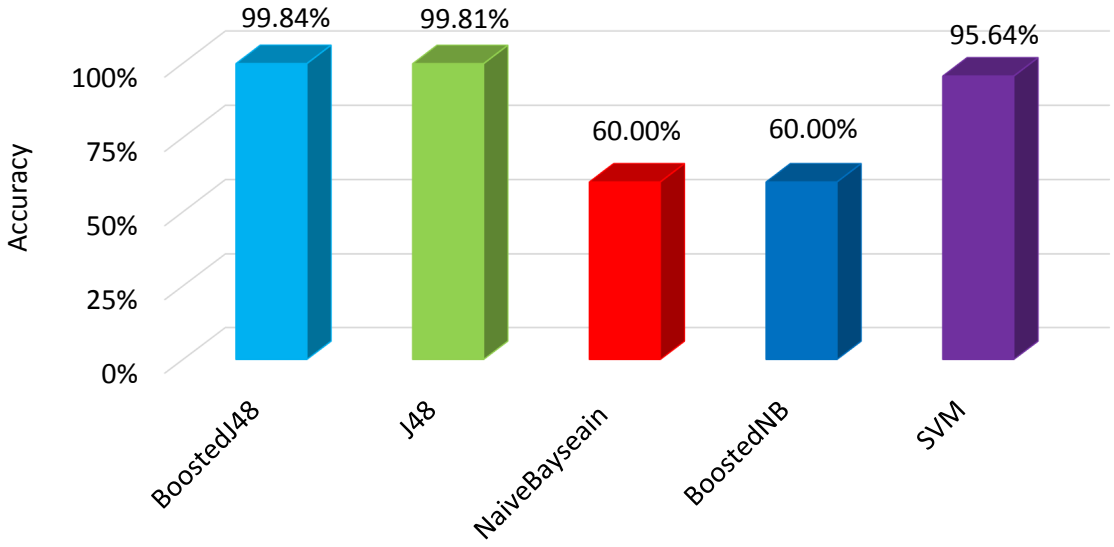


Figure 14: Accuracy Detection per Machine Learning Using Malicious and Benign (Private) Datasets

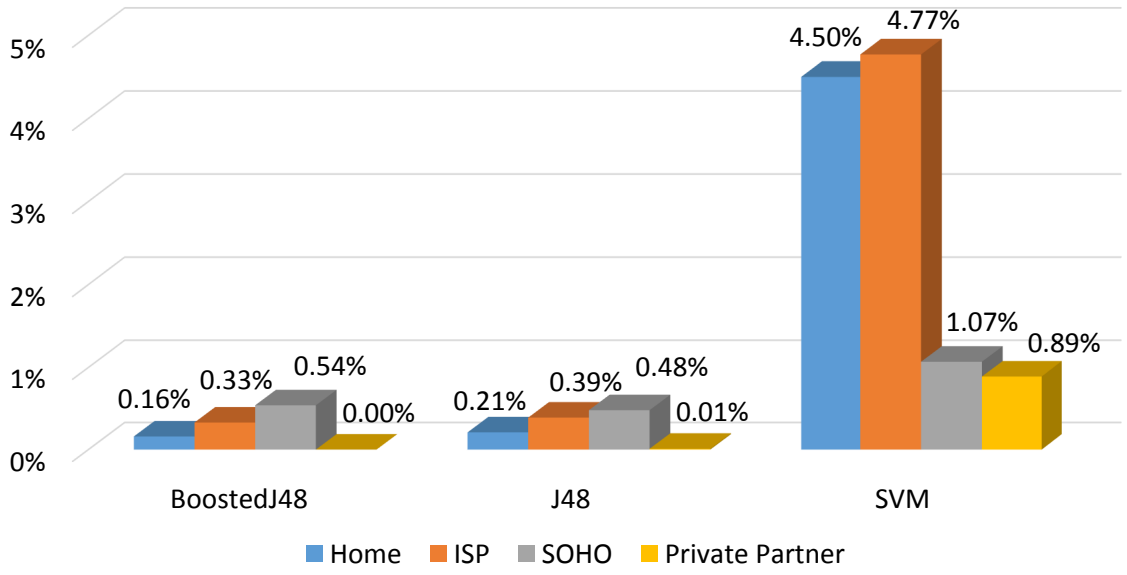


Figure 15: False Positive Rate per Machine Learning

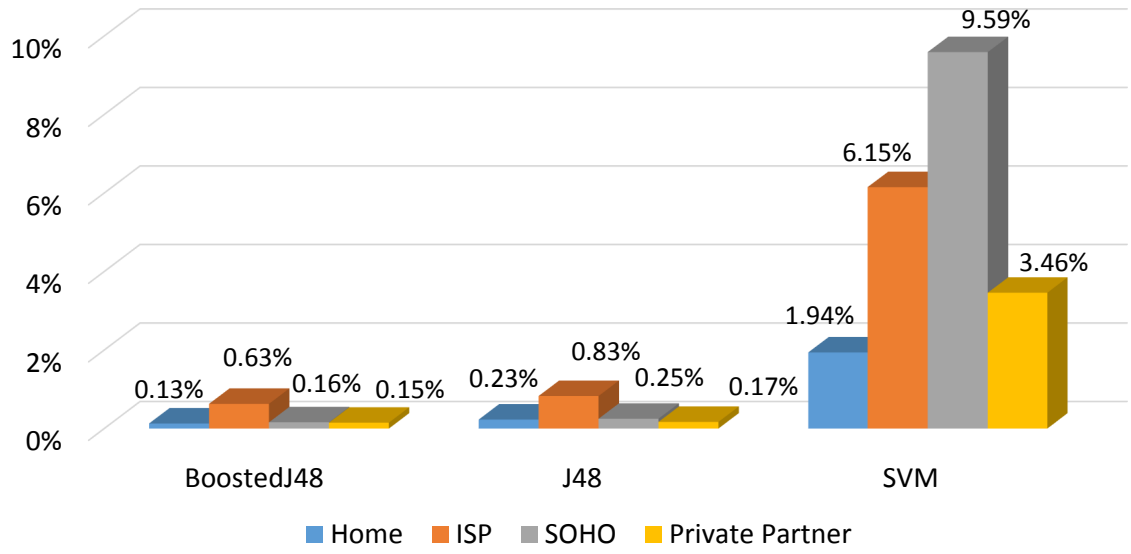


Figure 16: False Negative Rate per Machine Learning

Why J48 algorithm performed better than other machine learning algorithms?

Naïve Bayesian has shown bad results since it relies on independency of features, which is not the case in our study. For example, packet length depends on frame length. Regarding SVM, we used it with the default option where linear classification is performed. As such, it showed a problem with probabilities of class membership. J48 does not rely on features dependency and tends to perform better with limited number of classes, which is the case of our work since we have two classes. Moreover, after finding that J48 is the most suitable algorithm, we used 10 times cross-validation method to select the training and testing data. This is done to ensure that J48 algorithm maintains high accuracy and low false positive and negative rates even if the training and testing data change. Figures 17, 18, 19, and 20 summarize the performance of J48 algorithm in each data set by providing the accuracy and false positive and negative rates.

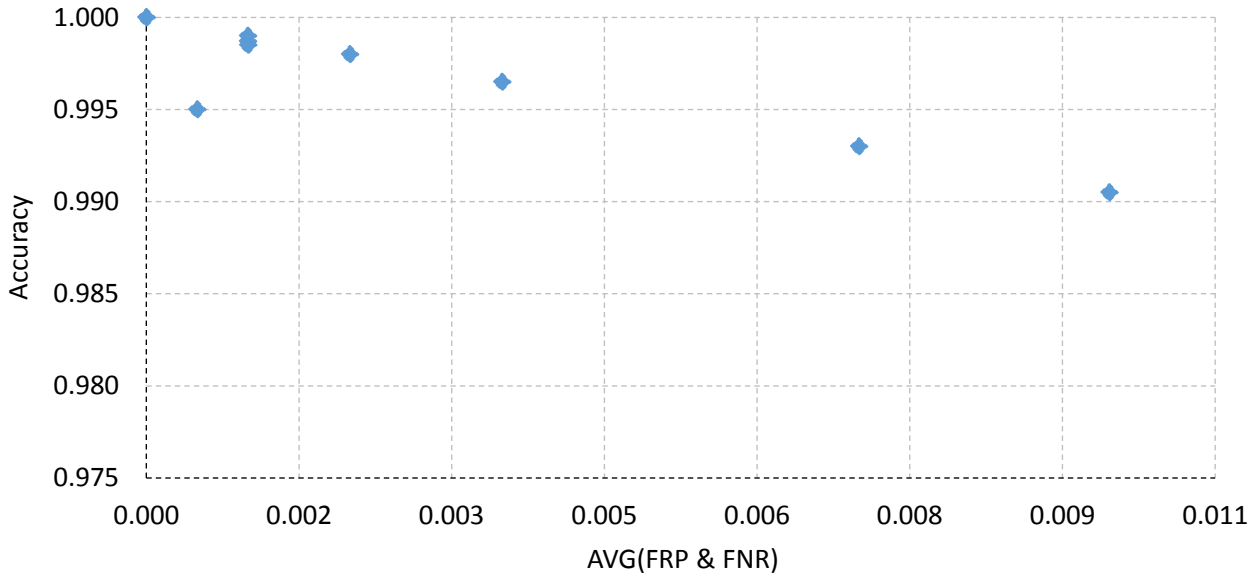


Figure 17: Scatter Plot for the J48 Classifier Detection Performance (DR VS AVG(FPR, FNR)) using Malicious and Benign Home datasets

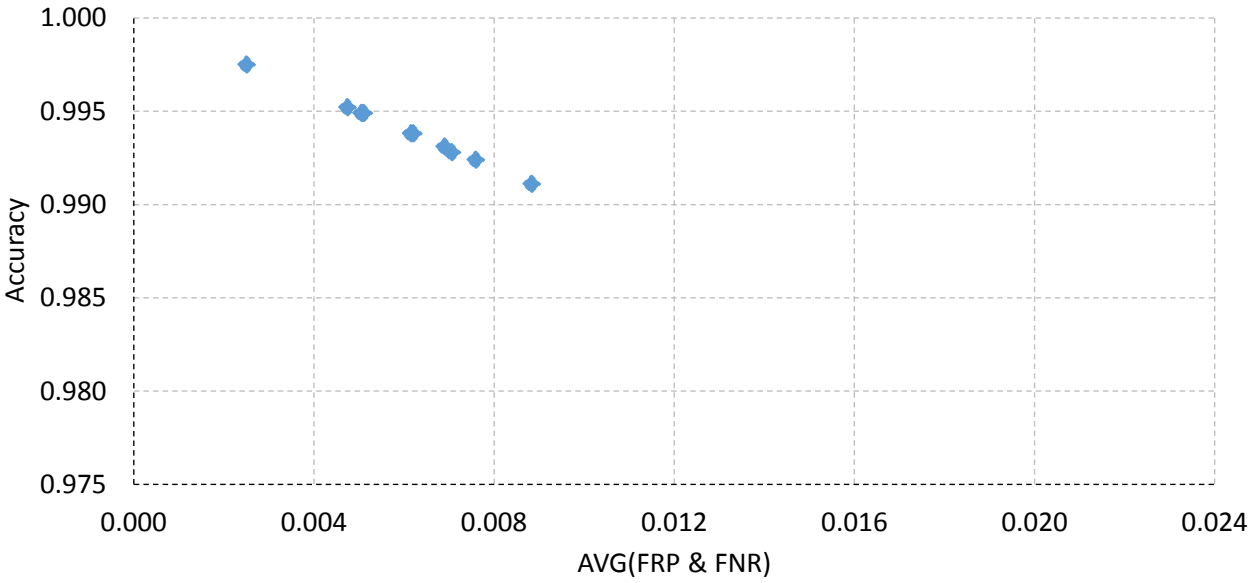


Figure 18: Scatter Plot for the J48 Classifier Detection Performance (DR VS AVG(FPR, FNR)) using Malicious and Benign ISP datasets

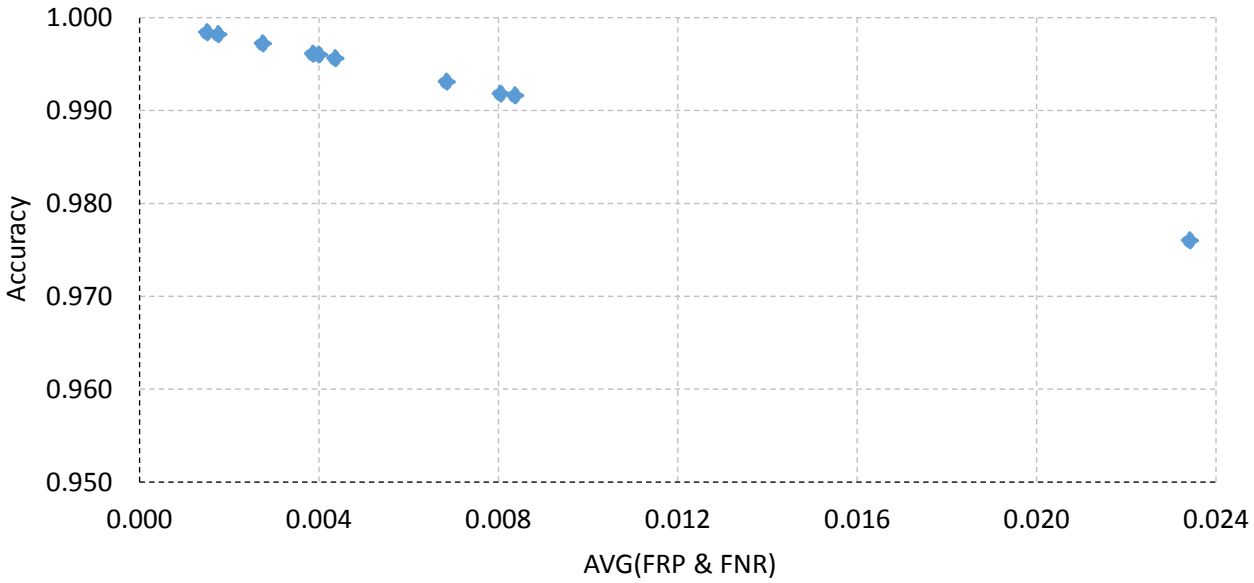


Figure 19: Scatter Plot for the J48 Classifier Detection Performance (DR VS AVG(FPR, FNR)) using Malicious and Benign SOHO datasets

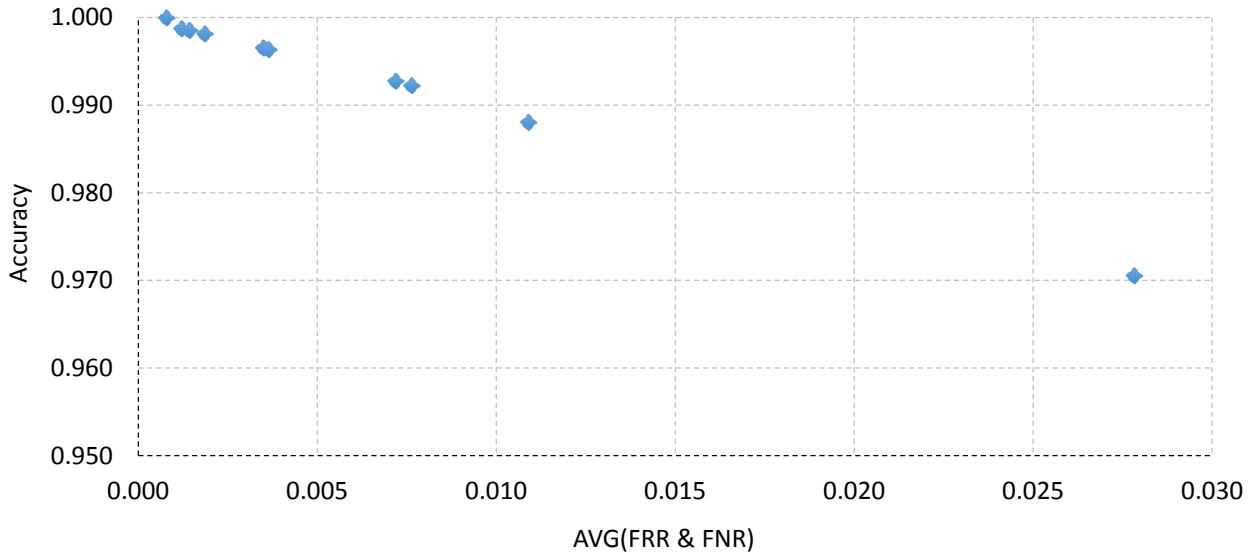


Figure 20: Scatter Plot for the J48 Classifier Detection Performance (DR VS AVG(FPR, FNR)) using Malicious and Benign Private datasets

What are the observations discovered in J48 decision trees?

Before digging into J48 decision models, we have loaded different datasets into parallel coordinates graphs. Such kind of graph is a good artifact to visualize high dimensional data and extract useful observations. Figures 21, 22, 23 and 24 illustrate how malicious data differs from benign in each dataset. Accordingly, by looking thoroughly into parallel graphs, we conclude the following observations:

- Forward and backward inter-arrival time values, duration of flow and number of forward packets and bytes are good indicators to distinguish malicious and benign traffic. As such, decision tree models generate decision rules, where the roots are usually the features that highly overlap between malicious datasets and benign ones. The distinctive features are mainly used as leaves to make final decisions on benign and malicious traffic.

- Since distinctive features are more noticeable to distinguish between malicious dataset Wisnet (Home), Wisnet (SOHO) and Private, it is easier to segregate between malicious dataset and Wisnet (Home), Wisnet (SOHO), Private then Wisnet (ISP). As a proof, the model generated for malicious and Wisnet (ISP) has more decision tree rules in comparison with the rest of the models. Table 4 illustrates the number of decision rules for each scenario.

Table 4: Decision Rules

Scenario	Number of Decision Rules
Malicious vs. WisNet (Home)	220
Malicious vs. WisNet (ISP)	1199
Malicious vs. WisNet (SOHO)	589
Malicious vs. Private	799

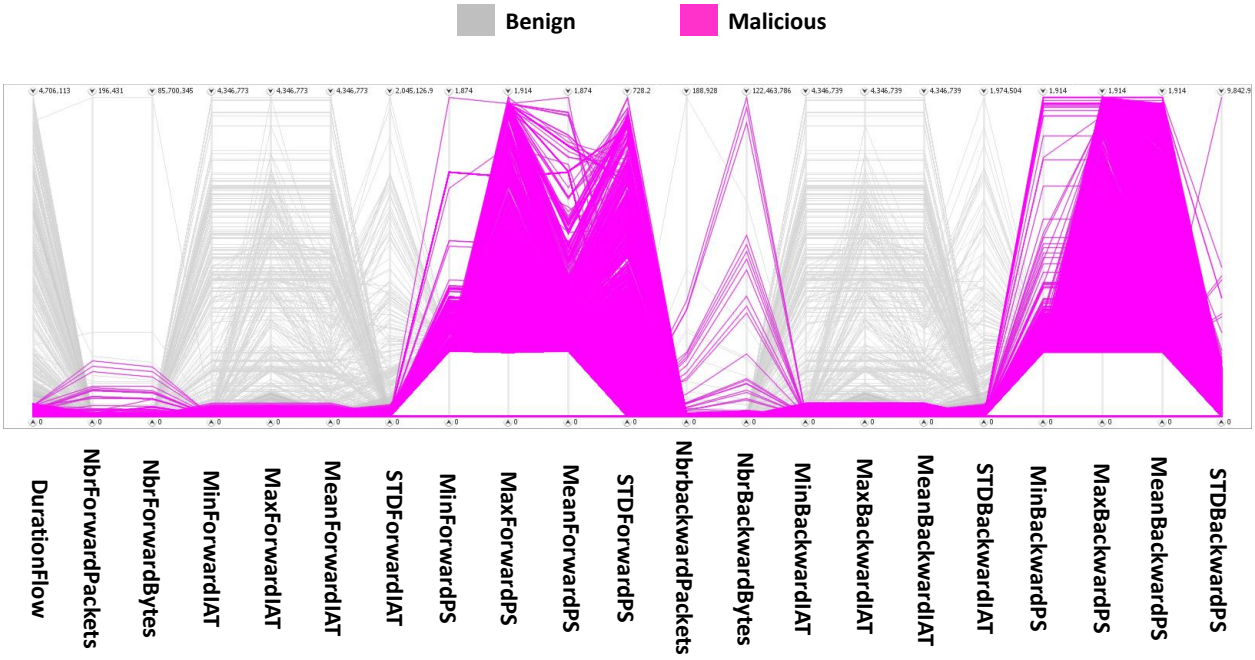


Figure 21: Distribution of the Malicious and Benign Home Traffic per Flow Features by Applying Parallel Graph

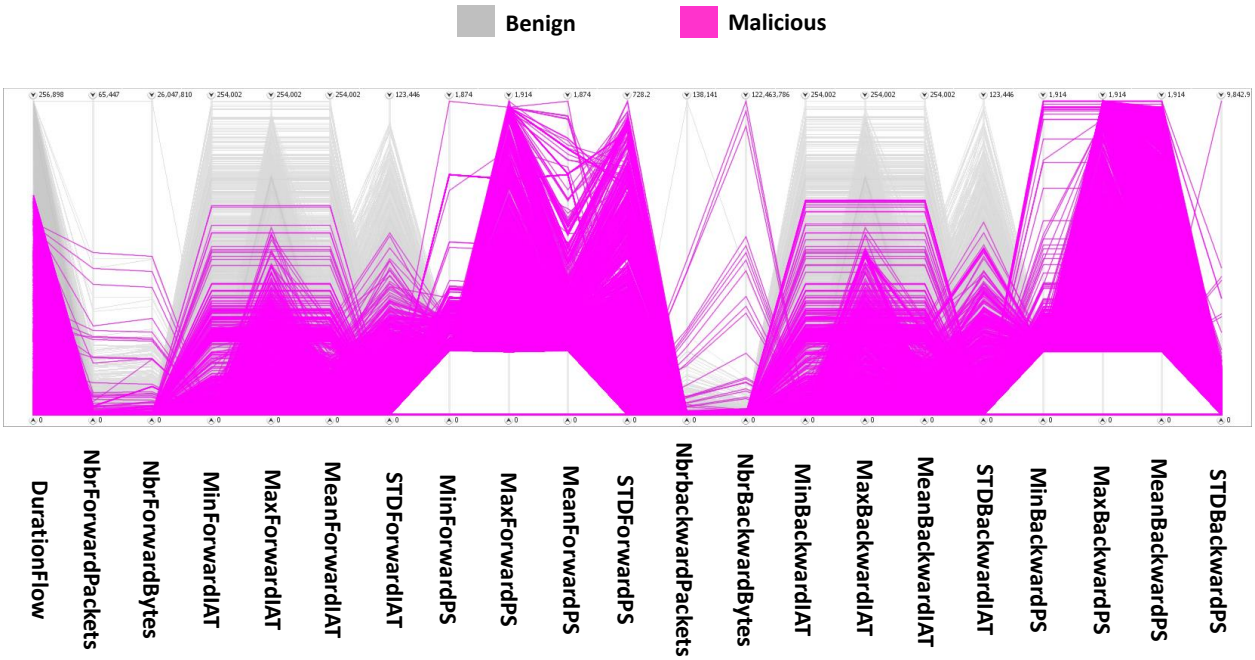


Figure 22: Distribution of the Malicious and Benign ISP Traffic per Flow Features by Applying Parallel Graph

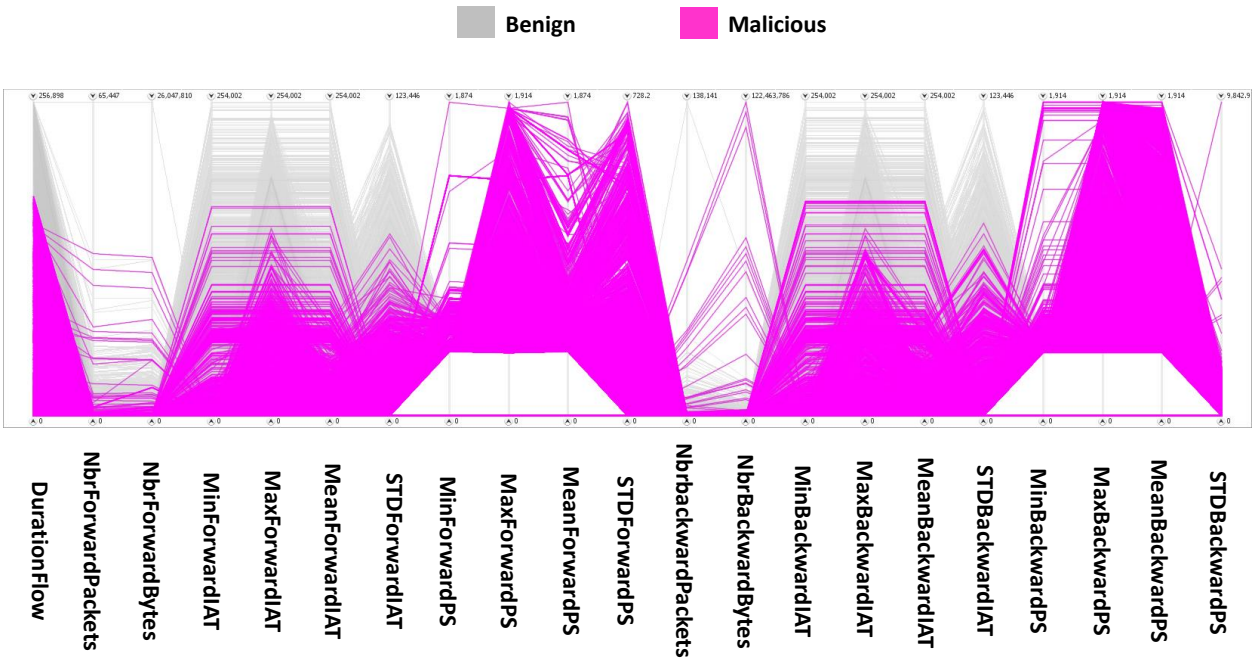


Figure 23: Distribution of the Malicious and Benign SOHO Traffic per Flow Features by Applying Parallel Graph

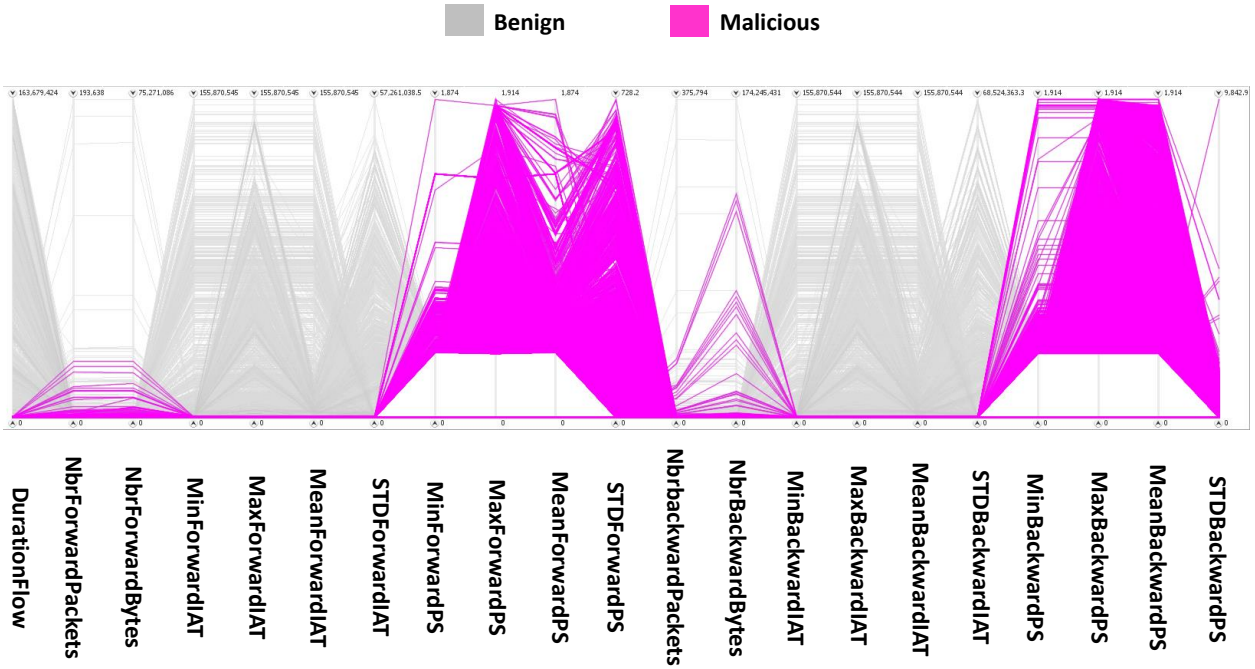
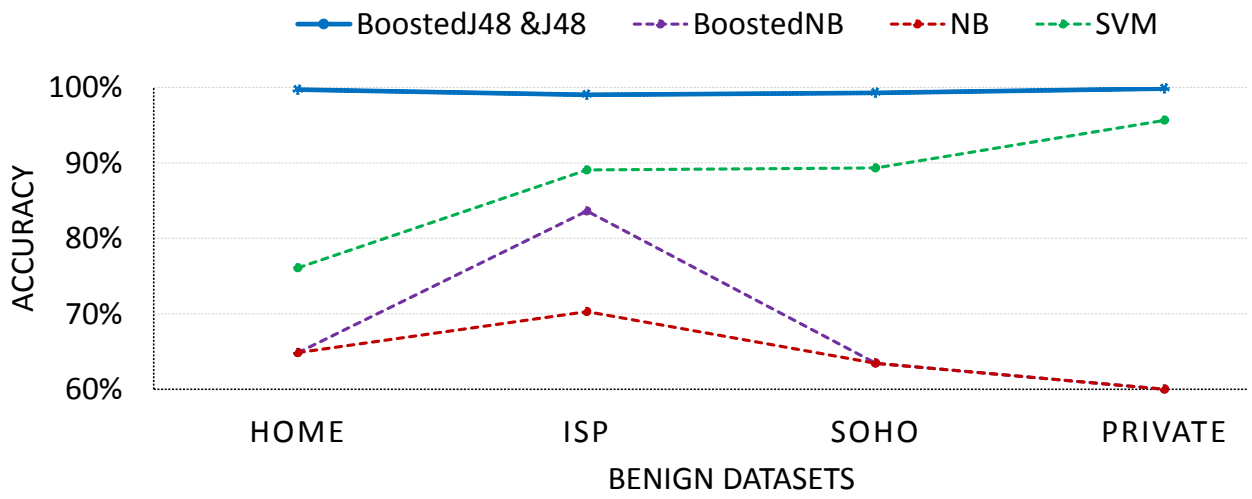


Figure 24: Distribution of the Malicious and Benign Private Traffic per Flow Features by Applying Parallel Graph

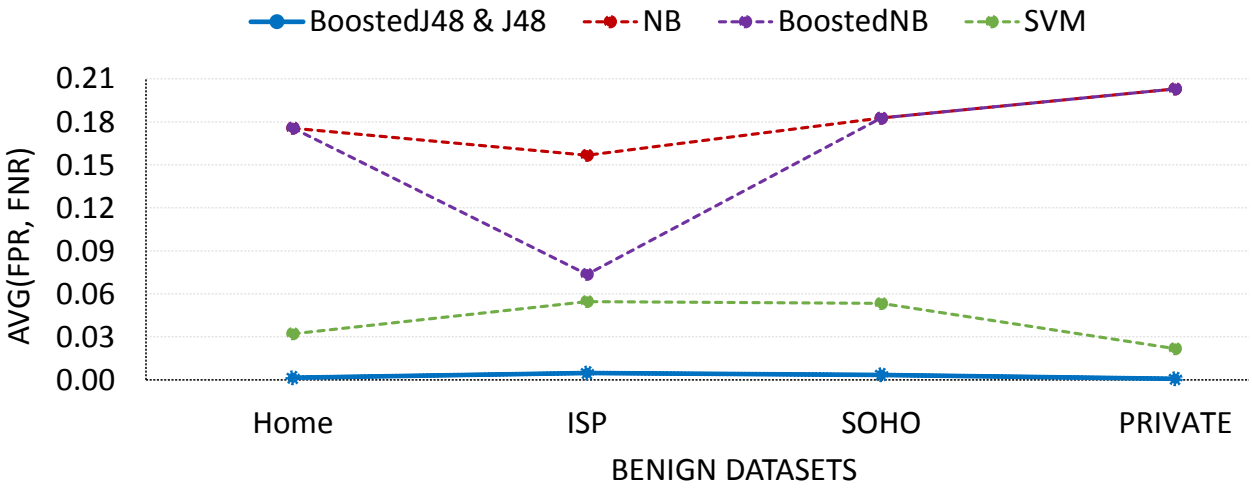
- Accordingly, flow duration, number of forward packets, bytes and inter-arrival times have small values for malicious traffic in comparison with traffic generated from a home network (e.g., Wisnet (HOME)), laboratory network (e.g., Wisnet (SOHO)) and private corporation network (e.g., private). For ISP network (Wisnet (ISP)), other features are needed since there is a big overlap between all features.

Can classification solution be generalized?

Boosted J48 and J48 algorithms have achieved high accuracy detection and low false positives and negatives in multiple datasets. The fact that we used different datasets has shown that J48 classification approach is robust since it maintained more than 98% accuracy and less than 0.006 average false alerts for each dataset, as illustrated in Figure 25a and Figure 25b respectively. Thus, these two algorithms allow for the segregation between malicious and benign traffic. Moreover, from the results, we conclude that our approach based on classifying the flow features can detect badness in different benign traffic traces with very high detection rate and low false alerts.



(a) Change of the Accuracy per Dataset



(b) Change of the Average of FPR and FNR per Dataset

Figure 25: Generalization Through Applying Detection Approach To Different Benign Datasets

3.8 Summary

In this chapter, we have presented a technique for the detection of malicious traffic. More precisely, we addressed the detection by using traffic classification. We exploited high level properties of flows to distinguish between malicious and benign traffic. We found that J48 algorithm is the most suitable classifier to fingerprint malicious traffic since it achieves 99% accuracy. Moreover, our approach can be generalized since it maintains more than 98% accuracy on different datasets. Our solution can be applicable to both clear-text and encrypted traffic to detect the malicious traffic since it is based on the high level flow information. In the next chapter, we describe how we attribute a detected malicious flow to the malware family that generate it.

Chapter 4

Malware Family Attribution

4.1 Introduction

Inferring the malware family, which is responsible for the generation of malicious communication, is of paramount importance from the mitigation standpoint. We create signatures for each malware family that capture their network behaviors. A signature consists of a sequence of network flows that are generated by the execution of the malware samples belonging to this family. Indeed, signatures are words over an alphabet that is made of labels. The latter are derived from a clustering exercise that amalgamates similar network flows within the same cluster. Consequently, the derivation of these signatures is achieved as follows:

- Clustering the malicious unidirectional flows in order to create the signature alphabet (Section 4.2).
- Generation of a communication sequence for each malware sample using the aforementioned signature alphabet. The obtained sequences are grouped per malware family (Section 4.3).

- Applying sequence mining techniques to group malware families that share same sequence patterns (Section 4.4).

- Generation of PDAs out of sequences representing groups of malware families. These PDAs act as generic network signatures for malware family groups (Section 4.5).

These steps together with the experimental results of the proposed method are explored in details in this chapter. Figure 26 illustrates the proposed method for malicious traffic attribution and prediction. It is important to mention that this approach relies on the detection method since it takes, as input, only the malicious flows.

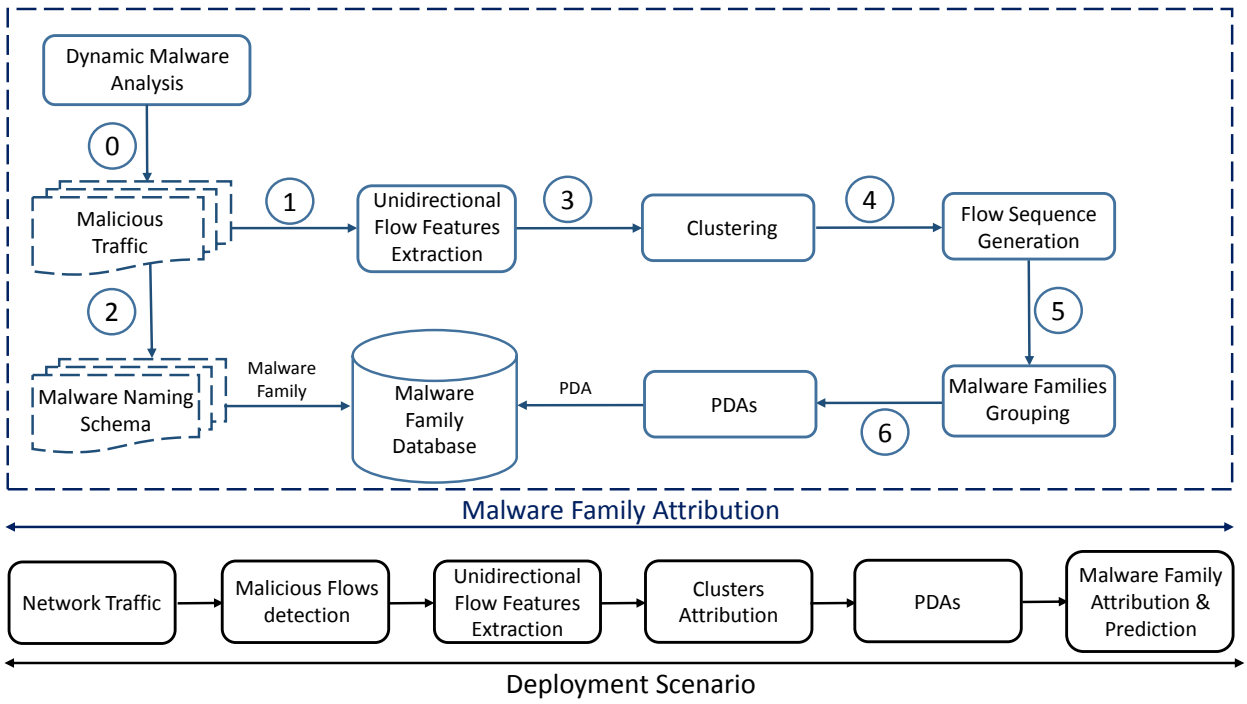


Figure 26: Overview of the Malicious Traffic Attribution and Prediction Framework

4.2 Malicious Flows Clustering

Malicious traffic tends to exhibit similar behaviors. As such, the term of co-expression is meant to identify common characteristics between malicious flows. In our work, we look for co-expressed unidirectional flows to group them into clusters. These clusters will be used later in malware family attribution as labels for malicious flows. The clustering techniques involve several steps: (1) Features extraction and selection, (2) Training the semi-supervised machine learning to produce the cluster solutions and (3) Measure the quality of the produced solution through evaluation criteria (internal and external similarity metrics). These steps are discussed in details in this section.

4.2.1 Unidirectional Flow Features

To characterize malicious traffic, we select a set of features that are derived from unidirectional flows. The difference with a bidirectional flow lies in the fact that IP addresses and ports are alternated according to the direction of communication. The reason behind using unidirectional flow features is that we want to preserve the semantics of inbound and outbound communication. Actually, the outbound traffic corresponds to those network flows that are generated by the execution of malware samples. On the other hand, the inbound traffic corresponds to the traffic received, as a response, from the Internet. We represent each flow by a vector of 45 features. This is achieved through a parser, which generates values that are stored in feature files, which are readable by CLUTO [11], a clustering toolkit. It is well-suited for clustering datasets arising in diverse research areas such as information retrieval [73], customer purchasing transactions [53], fraud detection [52], etc. Table 5 illustrates unidirectional flow features.

Table 5: Description of Unidirectional Flow Features

Features		
Generic Features	1	Total number of packets
	2	Flow duration
	3	Minimum inter-arrival time
	4	First quartile of inter-arrival times
	5	Median of inter-arrival times
	6	Mean of inter-arrival times
	7	Third quartile of inter-arrival times
	8	Maximum inter-arrival time
	9	Variance of inter-arrival times
	10	Minimum of control data size
	11	First quartile of control data size
	12	Median of control data size
	13	Mean of control data size
	14	Third quartile of control data size
	15	Maximum of control data size
	16	variance of control data size
	17	Total not empty packets
	18	Total packets size
Ethernet Layer Features	19	Minimum size in ethernet packets
	20	First quartile size in ethernet packets
	21	Median size in ethernet packets
	22	Mean size in ethernet packets
	23	Third quartile size in ethernet packets
	24	Maximum size in ethernet packets
	25	Variance size in ethernet packets
Network Layer Features	26	Minimum size in IP packets
	27	First quartile size in IP packets
	28	Median size in IP packets
	29	Mean size in IP packets
	30	Third quartile size in IP packets
	31	Maximum size in IP packets
	32	Variance size in IP packets
Transport Layer Features	33	Total ACK packets
	34	Total PUSH packets
	35	Total SYN packets
	36	Total FINE packets
	37	Total Urgent packets
	38	Total Urgent bytes
	39	Minimum TCP segment size
	40	Maximum TCP segment size
	41	Mean TCP segment size
	42	Minimum TCP window size
	43	Maximum TCP window size
	44	Mean TCP window size
	45	Total empty TCP window packet

4.2.2 Features Selection

Features selection is a method used to reduce the set of candidate features that leads to classification or clustering refinements. In our work, we estimate the quality of unidirectional flow features to proceed with their clustering. To do so, we consider SVM weights feature selection algorithm [23] that is integrated in RapidMiner [5]. By applying this algorithm, we reduce the number of features from 45 features to 25 features. The features are selected based on SVM complexity weighting factor. The features that are considered in our work are: Features 1 to 9 as well as features 16 and 18 from the generic features, features 19 to 25 from Ethernet layer features and features 26 to 32 from the network layer features.

4.2.3 Clustering

To cluster different inbound and outbound flows, we use the k-means Repeated Bi-Section (RBS) algorithm. The latter belongs to partitional clustering algorithms, which are considered as good artifacts for clustering large datasets since they have low computational requirements [13, 43].

K-means RBS: This algorithm uses a global criterion function [72], which derives the clustering solution. In our work, we use a parametric clustering method where the number of clusters is considered as an input. In this method, the solution is computed by repeating a sequence of bisections. The data is clustered initially into two groups, then each group is bisected until the criterion function is globally optimized and the desired number of clusters is found. The algorithm uses the vector space model [57] to represent each unidirectional flow. Each flow is represented by a dimension vector $fv = (f_1, f_2, ..., f_i)$, where f_i is the i^{th} unidirectional flow feature. To compute

similarity between vectors, we used cosine function [57] defined by Equation (7):

$$cos(v_i, v_j) = \frac{v_i^t v_j}{||v_i||||v_j||} \tag{7}$$

where: $||v_i||$: magnitude (square root) of vector i

$||v_j||$: magnitude (square root) of vector j

One of the intents of this work is to produce a clustering solution for inbound and outbound flows. A k-means clustering, based on a criterion function C, tends to find a solution such that C is optimized [72]. In our work, we use a hybrid function that is based on an internal function and an external function. The internal function tries to maximize the average pairwise similarities between flows that are assigned to each cluster. The internal function is given in Equation (8):

$$Maximize\ I = \sum_{r=1}^{k} \frac{||v_r||^2}{n_r} \tag{8}$$

Where: n_r: number of elements in a cluster r.

Unlike the internal criterion function, the external one derives the solution by optimizing a solution that is based on how the various clusters are different from each other (Equation (9)). The hybrid function combines external and internal functions to simultaneously optimize both of them (Equation (10)).

$$Minimize\ E = \sum_{r=1}^{k} n_r \frac{v_r^t v}{||v_r||} \tag{9}$$

Where: v: the composite vector of all vectors.

$$Maximize\, H = \frac{I}{E} \tag{10}$$

Based on K-means RBS algorithm, we create a set of experiments: inbound flow clustering solutions and outbound flow clustering solutions. The aim is to choose the best cluster solution based on the internal similarity metric (ISIM) and external similarity metric (ESIM). In the rest of this chapter, we refer to the clustering solution as the best out of many potential solutions.

4.3 Malware Flow Sequences Generation

Each Pcap trace is indexed by the corresponding malware sample hash. Algorithm 1 allows to create mappings between malicious Pcap traces and malware families. It collects these traces, which are indexed per malware sample hash and captures its family by Kaspersky malware name schema.

For each Pcap trace, we extract inbound/outbound flows from the indexed Pcap traces. These flows are supplied to CLUTO to generate clusters. Thus, each set of flows, composing a Pcap trace, is represented by a sequence of clustering labels. The sequences of flows are gathered iteratively and indexed per malware family. Algorithm 2 illustrates how we extract inbound/outbound flows from traces and how we gather iteratively the sequences and index them per malware family.

Algorithm 1: Malware Families Mapping with Network Traces

```
Data: Malicious Pcap traces P;
Result: Families mapping with Pcap traces F;
for i:=0;i < P.length();i++ do
    begin
        Trace P_i:=P.get(i);
        String hash:=P.getName();
        String family:=MalwareNameSchema(hash);
        if F.containsIndex(family) then
            F.getIndex(family).add(P_i);
        else
            F.setNewIndex(family);
            F.getIndex(family).add(P_i);
        end
    end
end
```

Algorithm 2: Mapping Malware Families With Flow Sequences

```
Data: Families mapping with Pcap traces F;
Result: Families Sequences Database S;
for i:=0;i < F.length();i++ do
    Map map:=F.get(i);
    String family:=F.getFamily();
    List P:=F.getTraces();
    List of sequences seq:=[ ];
    for j:=0;j < P.length();j++ do
        Trace P_j:=P.get(j);
        List flows:=P_j.ExtractOrderedFlows();
        for k:=0;k < flows.length();k++ do
            Flow flow_k:=flows.get(k);
            Sequence s:=identifyCluster(flow_k);
            seq.add(s);
        end
        if S.containsIndex(family) then
            S.getIndex(family).add(states);
        else
            S.setNewIndex(family);
            S.getIndex(family).add(states);
        end
    end
end
```

4.4 Malware Family Grouping

Since host-based information is discarded in our approach, it is challenging to segregate families that have the same network behavior. Accordingly, we group malware families into groups where the families belong to the same group only if they have the same network behavior. We use a sequence patterns mining technique, namely, PrefixSpan pattern growth algorithm [51], which identifies sequence patterns that represent malware families. Malware families that share sequence patterns are grouped together into a group.

Algorithm 3: Patterns Extraction

Data: Families Sequences Database S;
Minimum Support ms:=0.75;
Result: Mappings of families and patterns $maps$;
for $element$ *in* S **do**
    String $family$:=$element$.getFamily();
    Sequences s:=$element$.getSequences();
    Context $context$=LoadContext(s);
    Patterns p=PrefixSpan($context$,ms);
    Map map:=Map($family$,p);
    $maps$.add(map);
end

Algorithm 3 takes, as input, sequences of different families from the malware database and calls PrefixSpan pattern growth algorithm with a minimum support of 0.75 (a pattern "sub-sequence" is considered relevant if it has a frequency occurrence higher or equal to 75%) to compute frequent patterns. All families that have the same patterns are grouped into a malware family group. Each group is represented by the set of sequences belonging to malware families that share common patterns. Algorithm 4 illustrates how malware families are aggregated into groups.

Algorithm 4: Families Grouping

```
Data: Mappings of families and patterns maps;
Result: List of groups groups;
for map in maps do
    if groups.empty() then
        List of maps l:=[ ];
        l.add(map) Group g:=new Group(0,l);
        groups.add(g);
    else
        Patterns p1:=map.getPattern();
        flag:=True;
        j:=0;
        while flag do
            Group g:=groups.get(j);
            Map m:=g.getListMap();
            Patterns p2:=m.getPatterns();
            if p1==p2 then
                g.updateListMap(p1);
                flag=False;
            else
                j:=j++;
                if j:=groups.length() then
                    flag=False;
                    List of maps l:=[ ];
                    l.add(m);
                    Group newg:=new Group(0,l);
                    groups.add(newg);
                end
            end
        end
    end
end
```

4.5 PDAs Building

This module is responsible for the generation of PDAs and their mapping to groups of malware families. It takes, as input, a set of sequences that represent malware families belonging to the same group. Based on these sequences, a PDA is built to represent a malware group. A PDA is defined by a six-tuple as follows:

$PDA = < Q, \Sigma, \Gamma, \delta, s, F >$, where:

- Q: a finite set of states,
- Σ: a finite input alphabet (the different cluster classes),
- Γ: finite stack alphabet, s: start state, F: set of accepting states,
- δ transitions function: $Q \times \Sigma^* \times \Gamma^* \rightarrow Q \times \Gamma^*$

To build the PDA that is associated with a given malware family group, we use each sequence belonging to that group to construct one path in the PDA. The elements of each sequence are used as an input alphabet (Σ). After reading an input symbol, we create a transition to a new state in the PDA, where this symbol is pushed onto the stack. Hence, the move in the PDA is based on: input symbol, last symbol seen (top of the stack) and current state. Once all the elements of the sequence are processed and transitions are built, one path of the PDA is created. As such, a PDA is an aggregation of different sequences that represent a malware family group.

Figure 27 illustrates a PDA representing FakeInstaller Malware group. This automata has 15 states (omitting initial and finite states) and 21 possible transitions. For instance, the transition

between states I and $Q1$ occurs if the first flow belongs to a cluster labeled with $\omega 1$ and the stack is empty. Thus, $\omega 1$ is pushed onto the stack and the cursor is moved to state $Q1$. The next transition between states $Q1$ and $Q3$ occurs if the flow is labeled with $\xi 3$ and the stack contains $\omega 1$. As a result, $\omega 1$ is popped from the stack, $\xi 3$ is pushed onto the stack and the cursor is moved to state $Q3$. Iteratively, other transitions occur by checking the input symbol, the top of the stack and the current state. If state F is reached, then the FakeInstaller group signature is recognized.

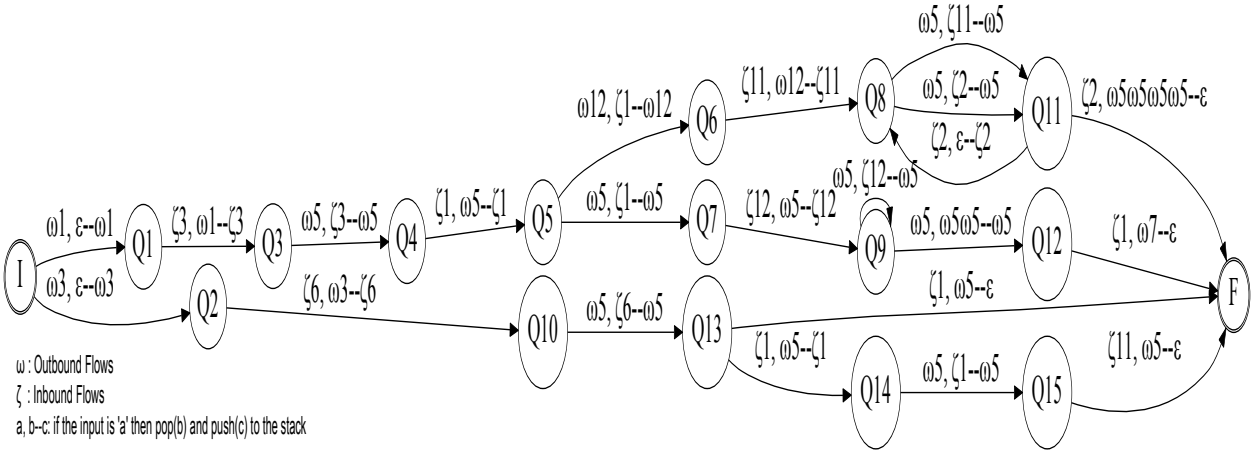


Figure 27: FakeInstaller Group PDA

4.6 Patterns Prediction

In order to attribute sequences of unidirectional flows, we need sequences that are interpretable by PDAs. However, if a sequence is not complete, it can be partially interpretable by PDAs. In order to cope with this problem, we corroborate the malware family attribution with a patterns prediction module. The latter is able to infer a set of malware families by computing the confidence level of patterns based on collected sequences. For each detected sequence, we check if it is

a starting sub-sequence of sequences that are mapped to malware families. For each malware family, we compute how many times the detected sequence appears as a starting sub-sequence. The confidence level represents the ratio of sub-sequence appearance and the total number of sequences for each malware family. Algorithm 5 illustrates how a list of mappings between malware families and confidence level are computed.

Algorithm 5: Patterns Prediction

```
Data: Families Sequences Database S;
Detected Sequence seq
Result: Maps of Malware Families and Confidence maps;
for element in S do
    String family:=element.getFamily();
    Sequences seqs:=element.getSequences();
    Integer cpt:=0;
    for s in seqs do
        if Subsequence(s,seq)==true then
            cpt++;
        end
    end
    Integer conf:=cpt/length(seqs);
    Map map:=Map(family,conf);
    maps.add(map);
end
```

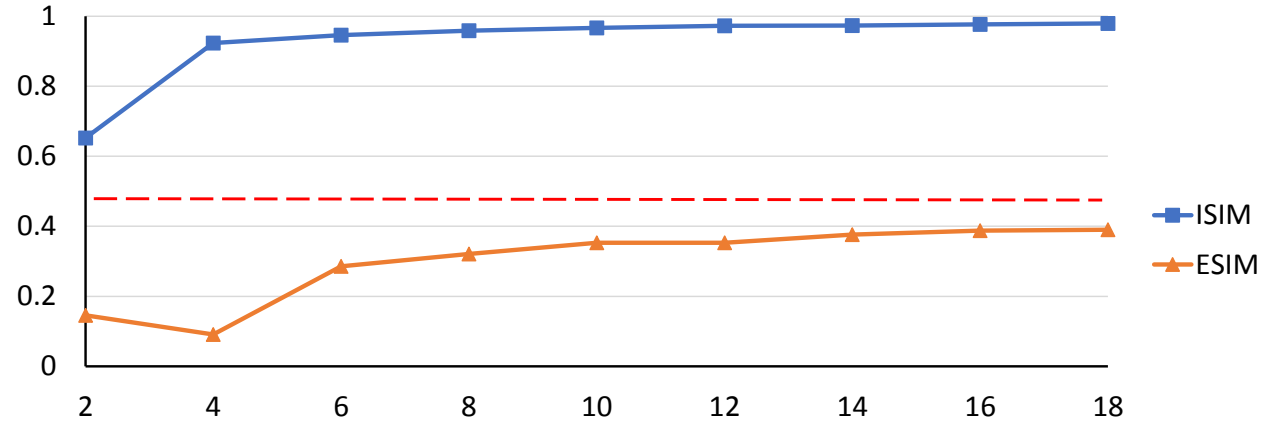
4.7 Experimental Results

The evaluation of malware family attribution falls into three parts: (1) Signature alphabet selection lies in describing different results obtained from clustering of unidirectional flows. (2) PDAs evaluation is based on the assessment of PDAs (signatures) attribution ability. (3) Malware families prediction depicts the ability to attribute malware families to uncompleted labeled sequences of unidirectional flows.

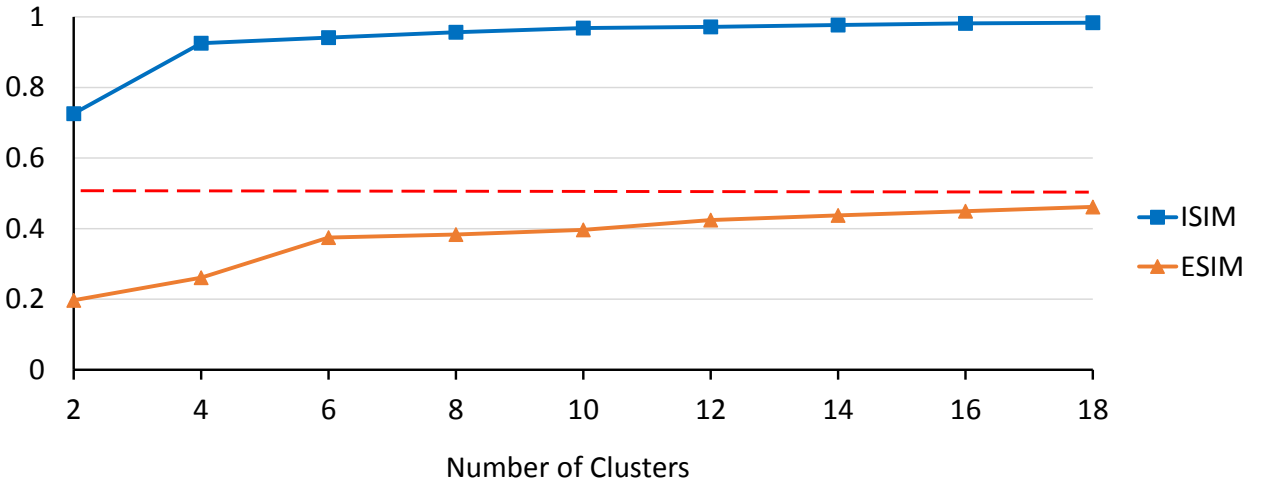
4.7.1 Malicious Traffic Clustering

To build different PDAs for malware families, we consider an alphabet that is represented by labels generated from K-means RBS clustering solutions for inbound traffic and outbound traffic. To evaluate such solutions, two metrics are considered: (1) High internal similarity metric (ISIM) average in all clusters, and (2) Moderate external similarity metric (ESIM) average in all clusters. In our case, it is hard to obtain a low external similarity average since we need a respectable number of clusters to represent PDAs. The clustering solutions vary from 2 to 18 clusters. We consider up to 18 clusters for both inbound and outbound flows to have the ability to potentially create a large number of PDAs that can differentiate between malware families. Figures 28a and 28b illustrate ISIM and ESIM averages for different clustering solutions for the inbound and outbound flows respectively. Regarding ISIM averages, we noticed that different solutions converged to high ratios starting from 4-way up to 18-way clustering solutions. Regarding ESIM averages, we wanted to maintain a moderate external similarity average between clusters. Thus, we set 0.5 as a ratio of tolerance. Finally, we chose solutions from 12-way up to 18-way clustering solutions since they

have high clusters cohesion (ISIM≥ 0.95) and moderate clusters isolation (ESIM≤ 0.5) as shown in Figures 28a and 28b.



(a) Inbound Flows Clustering

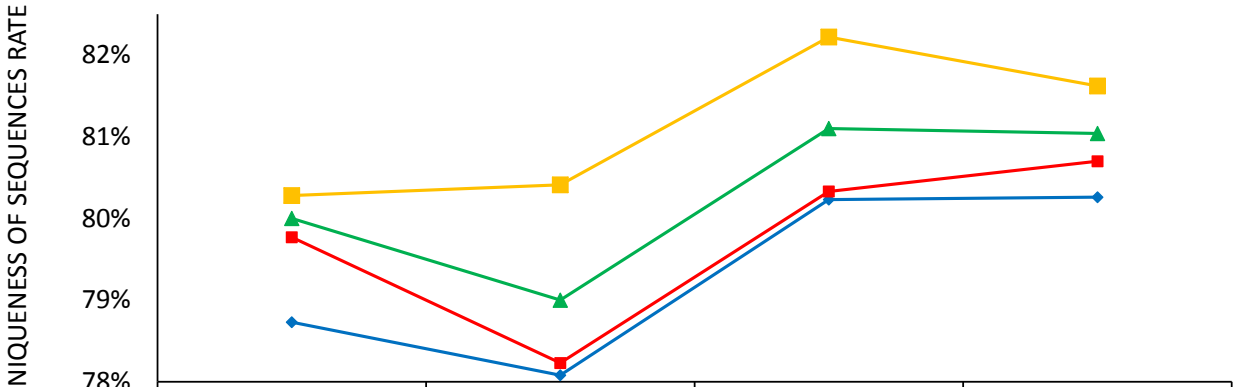


(b) Outbound Flows Clustering

Figure 28: Selecting Sequence Alphabet through Unsupervised Learning

To select the best solution among the 16 possible ones, we consider 4 additional scenarios. For each scenario, we fix the number of inbound flows clusters respectively to 12, 14, 16 and 18. Then, we vary iteratively the number of outbound flows clusters starting from 12, 14, 16 and 18. As a result, for each scenario, we have 4 combinations. Initially, we investigated the uniqueness of

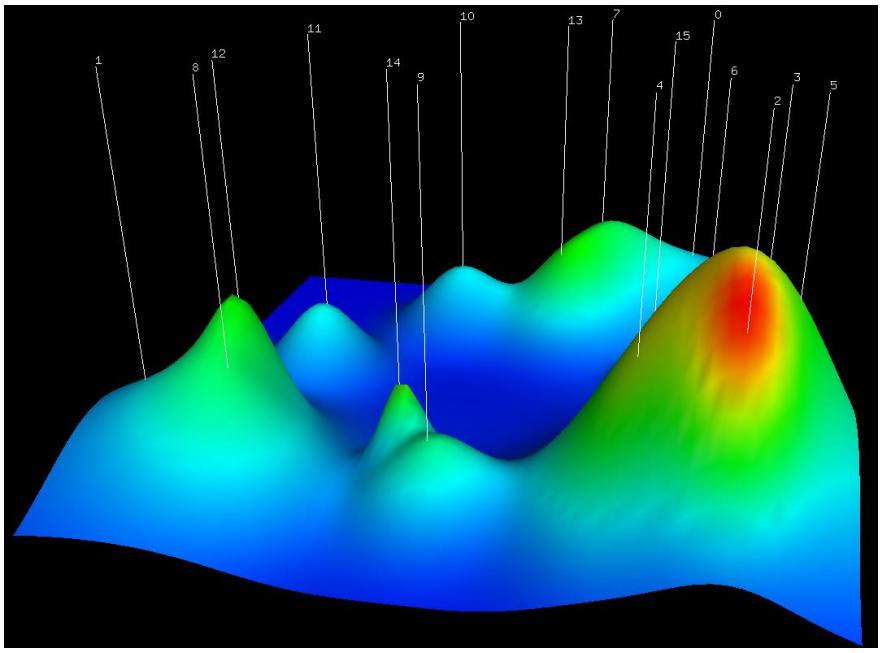
sequences ratio of clusters that are not shared by groups of malware families for each combination. The reason underlying this lies in the fact that the higher is the uniqueness of sequences ratio, the higher is the ability to segregate between groups of malware families.



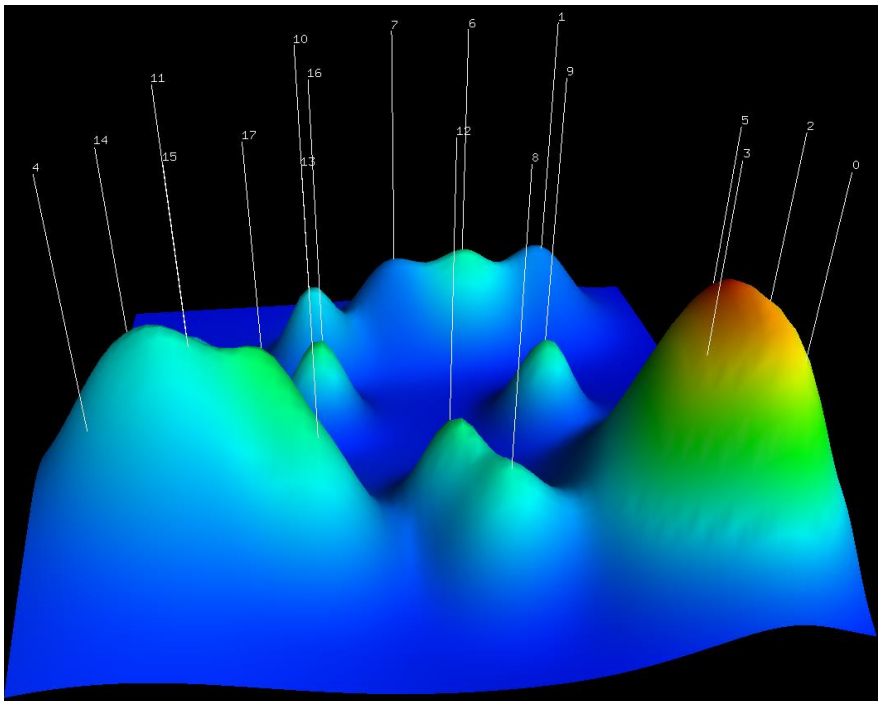
Outbound Clusters	12	14	16	18
12	78.73%	78.08%	80.23%	80.26%
14	79.77%	78.23%	80.33%	80.70%
16	80.00%	79.00%	81.10%	81.04%
18	80.28%	80.41%	82.22%	81.62%

Figure 29: Uniqueness of Sequences per Clustering Solution

The results are illustrated in Figure 29, where the highest ratio is 82.22%. It is achieved by 16 inbound clusters and 18 outbound clusters solution. By considering this solution, we obtained 128 groups of malware families. Figures 30a and 30b show the mountain views of the clustering solution. It gives an idea for the distribution and density of the malicious flow in each cluster. The two figures show the formation of 34 clusters. Hence, we obtained 34 alphabet elements to build PDAs.



(a) 16^{th} Inbound Clustering Solution



(b) 18^{th} Outbound Clustering Solution

Figure 30: Mountains Graphs of the Clustering Classes

4.7.2 Malware Family Attribution

To evaluate PDAs, we consider, for each malware family, 90% of malware samples to build PDAs and 10% for testing. As such, we set two metrics to evaluate PDAs, namely, uniqueness of sequences shared by groups and identification rate. By considering the first metric, we can assess the ability of PDAs to segregate between groups of malware families. In Figure 31, we illustrate uniqueness of sequences shared by groups. We observe that 82.22% of sequences are unique for all groups of malware families, where the rest of sequences (17.78%) are shared at most between 2 different groups. This means that for a testing malware there is a probability of 82.22% to be identified by one PDA.

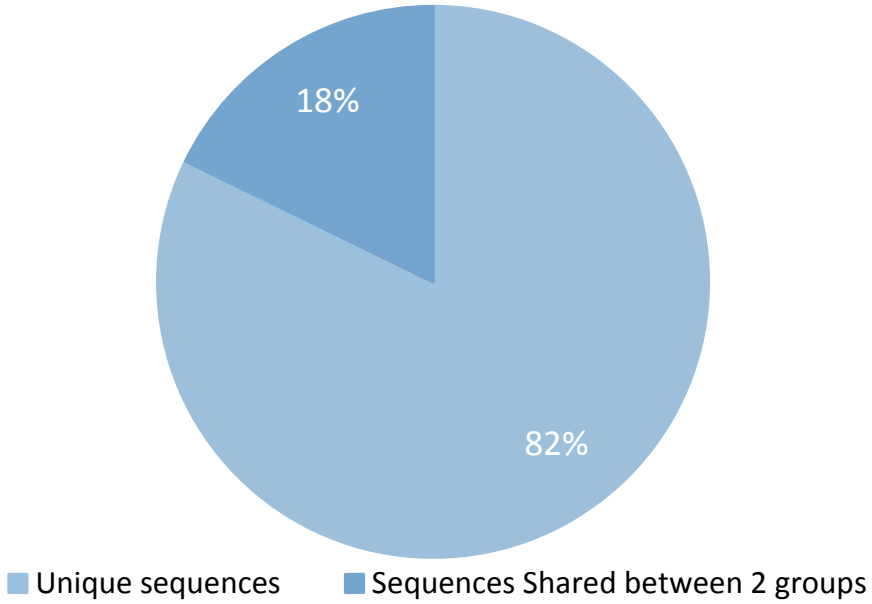


Figure 31: Evaluation of the Malware Family Attribution Through Shared Sequences

Regarding the identification rate, we use testing malware to see if they can be correctly identified by PDAs representing their groups. It is important to mention that the testing malicious traffic

Table 6: Detection Rate per Group

Uniqueness	**Groups**
100%	Group1(Crypt), Group2(Xtoober), Group4(Emud), Group5(Staem), Group6(Cinmus), Group7(Koobface), Group11(ICQ), Group13(Pex), Group9(CopyToWindows), Group10(Juched), Group8(KiayksayRen), Group12(Alipay), Group123(Gbod), Group14(Runouce), Group15(Fellow), Group16(LordSpy), Group17(Pioneer), Group18(Zwangi), Group61(RMS), Group21(Dybalom), Group22(ICQBomber), Group25(Guide), Group26(Servstar), Group29(Floder),Group30(Dyfuca), Group57(Kbot), Group31(Bagle), Group32(NetPass), Group33(SdBot), Group34(Kolabc), Group36(Tipp), Group37(FlyStudio), Group38(NSIS), Group39(Nilage), Group40(Induc), Group41(Downloader), Group42(Boaxxe), Group43(Alman), Group46(Artlu), Group49(VBInject), Group51(Tiny), Group55(Rebnip), Group52(SpyBot), Group59(Atua), Group84(RON), Group63(Gaba), Group64(SAMInside), Group67(AutoTsifiri), Group70(WinVNC), Group71(Boltolog), Group72(Gadu), Group83(Kate), Group73(Asper), Group74(Flystud), Group76(Wigon), Group87(Shiz), Group80(XPAntiSpyware), Group81(DigitalNames), Group85(MeSub), Group82(Koutodoor), Group86(FFAuto), Group93(Fullscreen), Group90(Fosniw), Group96(Vapsup), Group97(MediaGet), Group122(FC), Group98(Kuk), Group99(Koblu), Group100(Pher), Group101(Qvod), Group105(Banker), Group106(Skillis, SpectorPro), Group111(Yoshi), Group108(Proxyier, Simda), Group109(Nimnul), Group121(Cycler), Group118(ScreenSaver, Zango), Group128(Vivia), Group127(Papras) Group124(Joiner), Group125(Lpxenur), Group126(Otwycal).
$[85 - 100[\%$	Group3(AccPhish), Group77(Mabezat), Group79(Adload, Gbot, Daws, Banload, Dytka, EZula, ArchSMS, InstallCore, IRCNite, Look2Me, IRCNite, Plosa, Refroso, RivalGame, Swizzor, UBar), Group94(SCKeyLog, Zlob),Group102(Diple, Multi), Group113(FakeInstaller, PrivacyProtection).
$[70 - 85[\%$	Group19(Birele, Inject), Group47(Mydoom), Group66(Banito, Binder Prorat,Shakblades),Group89(Sohanad),Group107(Arto, Blat, Blocker, Clons, CodecPack, FenomenGame), Group116(LanFiltrator, PassRAR).
$[50 - 70[\%$	Group23(Skill), Group28(PcClient), Group44(Antilam, Bredolab, Foreign, Peed, Phires, Redirector, ShipUp, Tepfer, Upof, WBNA), Group48(Finlosky, Klone, Mahato,Mulwin), Group65(Midhos), Group56(FrauDrop, Geral), Group65(Midhos), Group91(Whimoo), Group112(iBryte, Spamer), Group115(Romeo).

is not used to generate PDAs. In Table 6, we illustrate the different ratio of identification (attribution) per group of malware families. Figure 32 shows that 75.21% of testing malware are correctly identified by their PDAs.

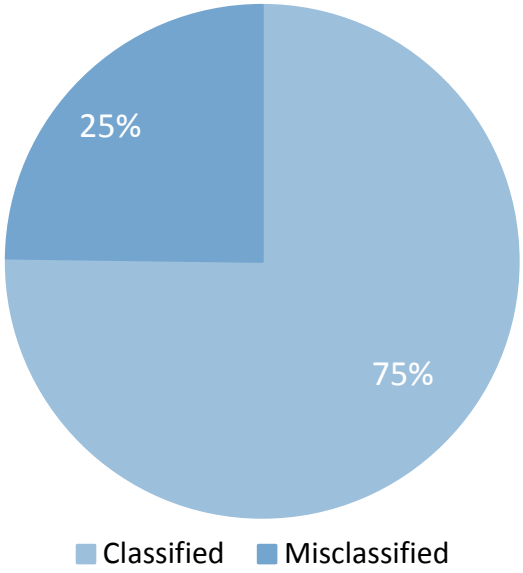


Figure 32: Malware Family Attribution Evaluation Through Detection Perfermance

4.7.3 Malware Family Prediction

Prediction of malware families or groups in the presence of incomplete sequences is important since it pinpoints to potential malware families that are responsible of the generation of malicious traffic. As such, we conduct an experiment where we vary the sub-sequences based on their length, then we compute the number of malware families and groups. Figure 33 shows the results obtained in this experiment. We observed that by getting a sub-sequence of 2 states, we can attribute up to 91 malware families and 27 malware groups. For a sub-sequence of 13 states or more, we can attribute to one malware family. For a sub-sequence of 6 states or more, we can attribute up to one malware group.

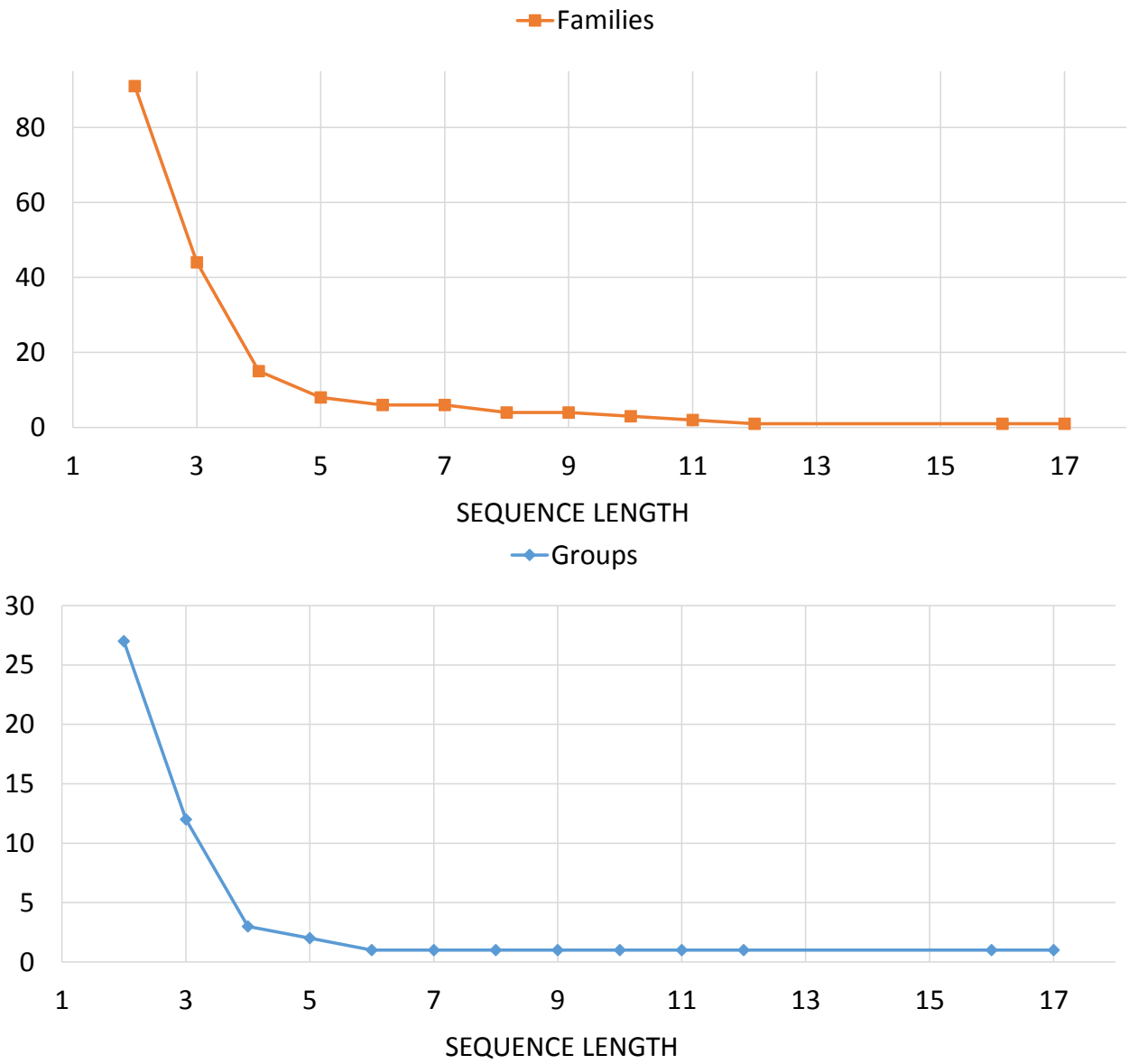


Figure 33: Number of Malware Families/Groups per Sequence Length

Figure 34 illustrates an example of a sub-sequence of 3 states and the corresponding families with the confidence levels. The latter represent the total number of flows that start with the sub-sequence among the total sequences belonging to each malware family. For instance, if we consider the sub-sequence *O2, I5, O17*, we obtain a confidence of 100% for *ICQ* malware family and a confidence of 1% for *Injector* malware family.

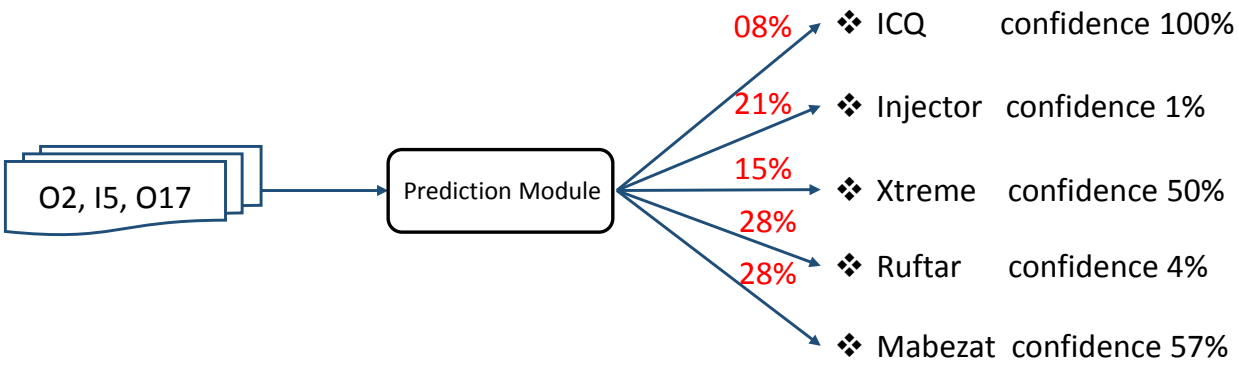


Figure 34: Prediction Output of a 3-state Sub-Sequence

4.8 Summary

In this chapter, we have used sequences of malicious labeled flows to segregate between malware family groups. To do so, we ran k-means RBS clustering algorithm for both inbound/outbound uni-directional flows and created co-expressed clusters for malicious traffic. We conducted clustering experiments by considering metrics mirroring the quality of different clustering solutions. We considered solutions with high internal similarity (ISIM) and moderate external similarity (ESIM). For each solution, we computed the uniqueness of sequences ratio. The optimal solution is 16 inbound clusters and 18 outbound clusters. This solution was considered to build sequences for malware families. To group malware families, we used PrefixSpan pattern growth algorithm to compute

frequent patterns among sequences of malware families. Malware families with the same frequent patterns are grouped into a malware family group. For each group, we used 90% of sequences of malware families to create a PDA, which characterizes the network behavior of malware families. The rest of sequences was used to test the identification ratio of PDAs. These PDAs are able to identify 75.21% of testing malware sequences.

Chapter 5

Conclusion

In this research initiative, we have presented a technique for the fingerprinting of malicious traffic and its attribution to malware families. Firstly, we addressed the detection by using traffic classification. We exploited high level properties of flows to distinguish between malicious and benign traffic. We found that J48 algorithm is the most suitable classifier to fingerprint malicious traffic since it achieves 99% accuracy. Moreover, our approach can be generalized since it maintains more than 98% accuracy on different datasets. Secondly, we used sequences of malicious labeled flows to segregate between malware family groups. To do so, we ran k-means RBS clustering algorithm for both inbound/outbound unidirectional flows and created co-expressed clusters for malicious traffic. We conducted clustering experiments by considering metrics mirroring the quality of different clustering solutions. We considered solutions with high internal similarity (ISIM) and moderate external similarity (ESIM). For each solution, we computed the uniqueness of sequences ratio. The optimal solution is 16 inbound clusters and 18 outbound clusters solution. This solution was considered to build sequences for malware families. To group malware

families, we used PrefixSpan pattern growth algorithm to compute frequent patterns among sequences of malware families. Malware families with the same frequent patterns are grouped into a malware family group. For each group, we used 90% sequences of malware families to create a PDA, which characterizes the network behavior of malware families. The rest of sequences was used to test the identification ratio of PDAs. These PDAs are able to identify 75.21% of testing malware sequences. In addition to the attribution of badness to malware families, we enhanced the proposed framework with a prediction module that attributes sub-sequences of malicious flows to a set of malware families/groups with confidence levels. Integrating a prediction ability to the framework is important during the deployment due to technical challenges in detecting complete sequences [19].

Regarding malware family attribution, as a future work, we plan to corroborate it with a non-deterministic modeling such as Hidden Markov Model (HMM). Such modeling allows to score the attribution of malicious flows sequences with likelihood measurement. This technique copes with the problem of attributing incomplete malicious flows sequence since the HMM model can be trained with sub-sequences (length greater or equal to 2). Last but not least, we aim to address the deployment of the framework.

Acknowledgements

The author would like to thank the Concordia security computer lab members namely: Amine Boukhtouta, Hamad Binsalleeh, Taher Azab, Son Dinh, Djedjiga Mouheb, Claude Fachkha, Elias Bou-Harb, Serguei A. Mokhov for their support, collaboration, encouragements and friendship.

Moreover, the author would like to acknowledge the help of ThreatTrack Security [4] company.

Bibliography

[1] Application layer packet classifier for linux. http://l7-filter.sourceforge.net/, available on Feb 15 2011, Visited on 11 March 2014.

[2] Class J48. http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html, visited on 11 March 2014.

[3] jNetPcap OpenSource. http://www.jnetpcap.com/, visited on March 11 2014.

[4] Malware Analysis with GFI SandBox (formerly CWSandbox). http://www.gfi.com/malware-analysis-tool, visited on 11 March 2014.

[5] RapidMiner. http://rapid-i.com/, visited on 11 March 2014.

[6] Service name and transport protocol port number registry. www.iana.org/assignments/port-numbers, last update March 10 2014, visited on March 11 2014.

[7] Snort Home Page. http://www.snort.org/, visited on March 11 2014.

[8] The Bro Network Security Monitor. http://www.bro.org/, visited on March 11 2014.

[9] Weka 3 Data Mining with Open Source Machine Learning Software in Java. http://www.cs.waikato.ac.nz/ml/weka/, visited on March 11 2014.

[10] Wireshark the world foremost network protocol analyzer. http://www.wireshark.org/, visited on March 11 2014.

[11] WISNET: Downloads. http://wisnet.seecs.nust.edu.pk/downloads.php, visited on March 11 2014.

[12] nDPI, 2012. www.ntop.org/products/ndpi/.

[13] Charu C. Aggarwal, Stephen C. Gates, and Philip S. Yu. On the merits of building categorization systems by supervised clustering. In Usama M. Fayyad, Surajit Chaudhuri, and David Madigan, editors, *KDD*, pages 352–356. ACM, 1999.

[14] Riyad Alshammari and A. Nur Zincir-Heywood. Can encrypted traffic be identified without port numbers, ip addresses and payload inspection? *Computer Networks*, 55(6):1326–1350, 2011.

[15] Tom Auld, Andrew W. Moore, and Stephen F. Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, 2007.

[16] F. Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05. USENIX Association, 2005.

[17] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*, CoNEXT '06, pages 1–12, New York, NY, USA, 2006. ACM.

[18] James R. Binkley and Suresh Singh. An algorithm for anomaly-based botnet detection. In *Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet - Volume 2*, SRUTI'06, pages 1–7, Berkeley, CA, USA, 2006. USENIX Association.

[19] Amine Boukhtouta, Nour-Eddine Lakhdari, Serguei A. Mokhov, and Mourad Debbabi. Towards fingerprinting malicious traffic. In *Proceedings of the 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013), Halifax, Nova Scotia, Canada, June 25-28, 2013*, pages 548–555, 2013.

[20] Peter Bright. Anonymous speaks: the inside story of the hbgary hack. `http://arstechnica.com/tech-policy/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack/`, available on Feb 15 2011, visited on 11 March 2014.

[21] S. Buehlmann and C. Liebchen. Joebox: a secure sandbox application for Windows to analyse the behaviour of malware, 2011. `http://www.joebox.org/`, visited on March 11 2014.

[22] Su Chang and Thomas E. Daniels. P2p botnet detection using behavior clustering & statistical tests. In *AISec*, AISec'09, pages 23–30, New York, NY, USA, 2009. ACM.

[23] Yin-Wen Chang and Chih-Jen Lin. Feature ranking using linear svm. *Journal of Machine Learning Research - Proceedings Track*, 3, 2008.

[24] Peter Cheeseman and John Stutz. Bayesian classification (autoclass): theory and results. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy,

editors, *Advances in knowledge discovery and data mining*, pages 153–180, Menlo Park, CA, USA, 1996. American Association for Artificial Intelligence.

[25] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 5–14. ACM, 2007.

[26] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: understanding, detecting, and disrupting botnets. In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, SRUTI'05, pages 1–6, Berkeley, CA, USA, 2005. USENIX Association.

[27] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[28] 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. ACM, 2008.

[29] Thomas Dullien, Rolf Rolles, and Ruhr universitaet Bochum. Graph-based comparison of executable objects. In *University of Technology in Florida*, 2005.

[30] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval.*, 64(9-12):1194–1213, October 2007.

[31] Yoav Freund. Boosting a weak learning algorithm by majority. *Inf. Comput.*, 121(2):256–285, September 1995.

[32] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 139–154, Berkeley, CA, USA, 2008. USENIX Association.

[33] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 1–16, Berkeley, CA, USA, 2007. USENIX Association.

[34] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS, 2008.

[35] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[36] Galen Hunt and Doug Brubacher. Detours: binary interception of win32 functions. In *Proceedings of the 3rd conference on USENIX Windows NT Symposium - Volume 3*, WINSYM'99, pages 14–14, 1999.

[37] Joonhyouk Jang, Sanghoon Choi, and Jiman Hong. A method for resilient graph-based comparison of executable objects. In *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, RACS '12, pages 288–289. ACM, 2012.

[38] Jian Kang, Jun-Yao Zhang, Qiang Li, and Zhuo Li. Detecting new p2p botnet with multi-chart cusum. In *NSWCTC - Volume 01*, NSWCTC'09, pages 688–691, Washington, DC, USA, 2009. IEEE Computer Society.

[39] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. In *HotBots*, HotBots'07, pages 1–7, Berkeley, CA, USA, 2007. USENIX Association.

[40] J. Zico Kolter and Marcus A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7:2006, 2006.

[41] Carsten Willems Konrad Rieck, Philipp Trinius and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.*, 19(4):639–668, December 2011.

[42] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *SIGCOMM Comput. Commun. Rev.*, 34(1):51–56, January 2004.

[43] Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *KDD*, KDD '99, pages 16–22, New York, NY, USA, 1999. ACM.

[44] Carl Livadas, Robert Walsh, David E. Lapsley, and W. Timothy Strayer. Using machine learning techniques to identify botnet traffic. In *LCN*, pages 967–974, Washington, DC, USA, 2006. IEEE Computer Society.

[45] Anthony Mcgregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In *In PAM*, pages 205–214, 2004.

[46] Andrew Moore, Michael Crogan, Andrew W. Moore, Queen Mary, Denis Zuev, Denis Zuev, and Michael L. Crogan. Discriminators for use in flow-based classification. Technical report, 2005.

[47] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. *SIGMETRICS Perform. Eval. Rev.*, 33(1):50–60, June 2005.

[48] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, SP'05, pages 226–241, Washington, DC, USA, 2005. IEEE Computer Society.

[49] Sang-Kyun Noh, Joo-Hyung Oh, Jae-Seo Lee, Bong-Nam Noh, and Hyun-Cheol Jeong. Detecting p2p botnets using a multi-phased flow model. In *ICDS*, ICDS'09, pages 247–253, Washington, DC, USA, 2009. IEEE Computer Society.

[50] Samir Ouchani, Otmane Ait'Mohamed, and Mourad Debbabi. A non-convex classifier support for abstraction-refinement framework. In *24th International Conference on Microelectronics (ICM), 2012*, pages 1–4, 2012.

[51] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. on Knowl. and Data Eng.*, 16(11):1424–1440, November 2004.

[52] Yi Peng, Gang Kou, A. Sabatka, Zhengxin Chen, D. Khazanchi, and Yong Shi. Application of clustering methods to health insurance fraud detection. In *Service Systems and Service Management, 2006 International Conference on*, volume 1, pages 116–120, 2006.

[53] Yi Peng, Gang Kou, Yong Shi, and Zhengxin Chen. Improving clustering analysis for credit card accounts classification. In *ICCS - Volume Part III*, ICCS'05, pages 548–553, Berlin, Heidelberg, 2005. Springer-Verlag.

[54] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[55] Costin Raiu. Cyber-threat evolution: the past year. *Computer Fraud & Security*, (3):5–8, 2012.

[56] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick G. Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Internet Measurement Conference*, pages 135–148, 2004.

[57] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[58] NORMAN SANDBOX. Norman SandBox Whitepaper, 2003. `http://download01.norman.no/whitepapers/whitepaper_Norman_SandBox.pdf`, available in 2013, visited on March 11 2014.

[59] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition, 2012.

[60] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. The earlybird system for real-time detection of unknown worms. Technical Report CS2003-0761, University of California, San Diego, August 2003.

[61] Stephen E. Smaha. Haystack: An intrusion detection system. *Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference*, page 37âĂŞ44, 1988.

[62] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. Botfinder: finding bots in network traffic without deep packet inspection. In *CoNEXT*, CoNEXT '12, pages 349–360, New York, NY, USA, 2012. ACM.

[63] A. Vasudevan and R. Yerraballi. *Sakthi: A Retargetable Dynamic Framework for Binary Instrumentation*. University of Texas at Arlington, 2003.

[64] A. Vasudevan and R. Yerraballi. Stealth breakpoints. In *In: 21st Annual Computer Security Applications Conference, 2005*, pages 381–392, 2005.

[65] A. Vasudevan and R. Yerraballi. Cobra: Fine-grained malware analysis using stealth localized-executions. In *In Proceedings of 2006 IEEE Symposium on Security and Privacy (Oakland 06)*, 2006.

[66] A. Vasudevan and R. Yerraballi. Spike: engineering malware analysis tools using unobtrusive binary-instrumentation. In *ACSC*, pages 311–320, 2006.

[67] Jaikumar Vijayan. Spamhaus hit by biggest-ever ddos attacks. `http://news.techworld.com/security/3437612/spamhaus-hit-by-biggest-ever-ddos-attacks/`, available on March 27 2013, visited on March 11 2014.

[68] Zheng Wang, Ken Pierce, and Scott McFarling. Bmat - a binary matching tool for stale profile propagation. *The Journal of Instruction-Level Parallelism*, 2:2000, 2002.

[69] J-Y. Xu, A. H. Sung, P. Chavez, and S. Mukkamala. Polymorphic malicious executable scanner by api sequence analysis. In *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems*, HIS '04, pages 378–383. IEEE Computer Society, 2004.

[70] Sebastian Zander. Netmate meter. `http://sourceforge.net/projects/netmate-meter/`, available on March 11 2014.

[71] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Automated traffic classification and application identification using machine learning. In *Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, LCN '05, pages 250–257, Washington, DC, USA, 2005. IEEE Computer Society.

[72] Ying Zhao and George Karypis. Criterion functions for document clustering: Experiments and analysis. Technical report, University of Minnesota, 2002.

[73] Shi Zhong and Joydeep Ghosh. Generative model-based document clustering: a comparative study. *Knowledge and Information Systems*, 8(3):374–384, 2005.