Exploiting Tactics, Techniques, and Procedures for Malware Detection

Yashovardhan Sharma

Keble College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2023

It is the time you spent on your rose that makes your rose so important.

— Antoine de Saint-Exupéry, *The Little Prince*

Acknowledgements

First and foremost, I would like to thank my family. My mom and dad, who are my rocks, without whom I would scarcely have the courage to take on the world. You are the reason I have managed to achieve anything in life. My brothers, without whom life would have been infinitely more dull and less enjoyable. It is through all of your collective support that I have made it to the end of this journey.

I would not be here without the constant love and support of my partner Eleonora. You have been a ray of light on the darkest of days, thank you. I am also grateful to the Giunchiglia Massa family for giving me another family away from home. Enrico in particular, whose guidance and advice throughout the years has made him a supervisor in his own right.

My friends, who have made my DPhil journey an immeasurably more joyful experience—Christina, Ayush, Namrata, Parth, Nikhil, Pratyush, Anjuli, Inda, Seb, Ani, Fatima, Matt, Harshita, Charu, Ralph, Ismail, Dimitra, Marc, Kostas, Amy, and Thiago—to name just a few. Your camaraderie and encouragement have sustained me through the highs and lows of this adventure.

My zest for knowledge is a product of several excellent institutions and some brilliant teachers within them. It is thanks to places like Rhymes, Mira Nursery, the Mother's International School, and IIIT-Delhi that I am who I am today. In particular, I would also like to express my gratitude to some wonderful teachers who fundamentally altered my learning experience—Jayshree ma'am for being the most supportive teacher one could ask for, Sumita ma'am for letting an 8-year old read Satyajit Ray even though some thought I was "too young" to understand them, Shaily ma'am for inculcating a love for maths that persists till today, Renu ma'am for encouraging the creativity in my writing (your immense pride and sole remark of "Lovely!" on my first creative writing essay about a boy who imagined rainbows were created by a flock of birds leaving behind a trail of colours, gave me the license to write and express myself freely), Deepti ma'am for further encouraging and believing in my mathematical abilities, Ruchi ma'am for being the best class teacher one could ask for, Benu ma'am for being unabashedly proud and encouraging of my writing abilities, Swarupa ma’am for being extremely kind and supportive at any and all points in time, Tarang ma’am for simultaneously being the hardest taskmaster and the most fun teacher I have had the pleasure of knowing, Shweta ma’am for always being supportive of my budding computer science skills and allowing me to thrive both in her lab and her classroom, Prof. Vinayak for introducing me to the world of networks and being a wonderful mentor through the years, Prof. Somitra for sparking my love for security through his excellent Applied Cryptography course and his teaching in general, and lastly the late Prof. Raj Ayyar who ignited and helped sustain my love for philosophy and critical thinking, and whose wit and charm shall forever be missed; you all have made me a better and far more knowledgeable human being.

Last but not least, I would like to thank my supervisor Ivan, without whose support I could not have made it through this DPhil. I would also like to thank Simon for his guidance and advice in trying times, such as when dealing with the infamous Reviewer 2. I have also had the pleasure of working and interacting with some wonderful people in the Robert Hooke Building; your company shall be sorely missed. From an administrative perspective, my eternal gratitude goes out to Janet, Jordan, and the Keble College Office for being the only entities in the entire university to always respond on time, and for caring about the individuals they were responsible for, above and beyond the ambit of their professional obligations. You cannot imagine the difference your kindness and care made.

To all those who were obstacles in my path, thank you—you only made me stronger, more determined, and wiser. The brick walls are there for a reason, and they certainly did not stop me.

We have Palaeolithic emotions, medieval institutions,
and godlike technology.

— Edward O. Wilson

Abstract

There has been a meteoric rise in the use of malware to perpetrate cybercrime and more generally, serve the interests of malicious actors. As a result, malware has evolved both in terms of its sheer variety and sophistication. There is hence a need for developing effective malware detection systems to counter this surge. Typically, most such systems nowadays are purely data-driven—they utilise Machine Learning (ML) based approaches which rely on large volumes of data, to spot patterns, detect anomalies, and thus detect malware. In this thesis, we propose a methodology for malware detection on networks that combines human domain knowledge with conventional malware detection approaches to more effectively identify, reason about, and be resilient to malware. Specifically, we use domain knowledge in the form of the Tactics, Techniques, and Procedures (TTPs) described in the MITRE ATT&CK ontology of adversarial behaviour to build Network Intrusion Detection Systems (NIDS). Through the course of our research, we design and evaluate the first such NIDS that can effectively exploit TTPs for the purpose of malware detection. We then attempt to expand the scope of usability of these TTPs to systems other than our specialised NIDS, and develop a methodology that lets any generic ML-based NIDS exploit these TTPs as model features. We further expand and generalise our approach by modelling it as a multi-label classification problem, which enables us to: (*i*) detect malware more precisely on the basis of individual TTPs, and (*ii*) identify the malicious usage of uncommon or rarely-used TTPs. Throughout all our experiments, we rigorously evaluate all our systems on several metrics using large datasets of real-world malware and benign samples. We empirically demonstrate the usefulness of TTPs in the malware detection process, the benefits of a TTP-based approach in reasoning about malware and responding to various challenging conditions, and the overall robustness of our systems to adversarial attack. As a consequence, we establish and improve the state-of-the-art when it comes to detecting network-based malware using TTP-based information. This thesis overall represents a step forward in building automated systems that combine purely-data driven approaches with human expertise in the field of malware analysis.

Publications

The contents of this thesis are based upon the following publications:

1. Yashovardhan Sharma, Simon Birnbach, and Ivan Martinovic. “RADAR: A TTP-based Extensible, Explainable, and Effective System for Network Traffic Analysis and Malware Detection”. *European Interdisciplinary Cybersecurity Conference (EICC)*, 2023.

2. Yashovardhan Sharma, Eleonora Giunchiglia, Simon Birnbach and Ivan Martinovic, “To TTP or not to TTP?: Exploiting TTPs to Improve ML-based Malware Detection”. *IEEE International Conference on Cyber Security and Resilience (CSR)*, 2023.

3. Yashovardhan Sharma, Simon Birnbach, and Ivan Martinovic. “Exploiting TTPs to Design an Extensible and Explainable Malware Detection System” *Journal of Universal Computer Science (JUCS)*, 2024.

4. Yashovardhan Sharma and Ivan Martinovic. “A TTP by TTP Approach: Precise Malware Detection via Malicious TTP Recognition”. (*Under submission*).

While all publications are joint work with others, I have only included published work where I was the first author. As such, I was the main contributor to the projects, although I had help from my co-authors in conducting the experiments in [2] and preparing the publications.

Research work related to the topic of my thesis is currently ongoing, under a grant from the European Space Agency (ESA) that I am co-leading with my supervisor. Given that the results of this research project have not been finalised yet, this work has been excluded from the purview of this thesis.

Contents

List of Figures **ix**

List of Abbreviations **xi**

1 Introduction **1**

1.1 Motivation 1

1.2 Scope of this Thesis 4

1.3 Ethical Considerations 4

1.4 Outline 5

2 Literature Review **7**

2.1 Introduction 7

2.2 The MITRE ATT&CK Framework 8

2.2.1 Background 8

2.2.2 TTP Usage in Cyber Security Research 10

2.3 Malware Detection 12

2.3.1 Content-based Approaches 12

2.3.2 Behaviour-based Approaches 19

2.3.3 Machine Learning based Approaches 29

2.3.4 Graph-based Approaches 36

2.4 Malware Classification 38

3 Extensible and Explainable Malware Detection using TTPs **42**

3.1 Introduction 42

3.2 System design 45

3.2.1 Data Ingestion and Graph Construction 46

3.2.2 TTP Detection 53

3.2.3 TTP-based Classification System 55

3.3 Evaluation 58

3.3.1 TTP Extraction 59

3.3.2 Classification performance 61

3.4 Discussion 67

3.5 Related Work 70

3.6 Limitations . . . 71
3.7 Conclusions . . . 71

4 A Black-Box Approach for Exploiting TTPs for Malware Detection 73
4.1 Foreword . . . 73
4.2 Introduction . . . 74
4.3 Methodology . . . 77
4.3.1 Network Flows . . . 78
4.3.2 TTP Annotation of Flows . . . 78
4.3.3 Bag of Flows . . . 80
4.3.4 Classification . . . 83
4.4 Evaluation . . . 85
4.4.1 Dataset . . . 85
4.4.2 Metrics . . . 86
4.4.3 Detailed Comparison and Statistical Significance . . . 87
4.4.4 Robustness Under Data Scarcity . . . 89
4.4.5 Robustness to Bin Size Variation . . . 91
4.4.6 Robustness to Injection of Benign Flows . . . 93
4.4.7 Overall Comparison . . . 96
4.5 Discussion . . . 96
4.6 Conclusions . . . 97

5 Precise Malware Detection using TTPs 98
5.1 Foreword . . . 98
5.2 Introduction . . . 99
5.3 System Architecture . . . 103
5.3.1 Constructing the Bag of Flows . . . 104
5.3.2 Multi-label DNN Classification . . . 106
5.4 Evaluation . . . 109
5.4.1 Datasets . . . 110
5.4.2 Metrics . . . 113
5.4.3 Evaluating Malicious Usage of TTPs . . . 116
5.4.4 Evaluating Sample Detection . . . 119
5.5 Experimental Analysis . . . 121
5.5.1 Threshold Variation per TTP . . . 122
5.5.2 Data Scarcity . . . 123
5.5.3 Adversarial Benign Flow Injection . . . 127
5.6 Discussion . . . 130
5.7 Related Work . . . 134
5.8 Conclusions . . . 135

6 Summary and Future Work **136**

6.1 Summary 136

6.2 Datasets 138

6.3 Future Work 138

Bibliography **141**

List of Figures

3.1 RADAR system overview. 45
3.2 Graph schema for network traffic. 50
3.3 Distribution of flows with a varying threshold of packet outflow/inflow ratio, with a logarithmic Y axis. 54
3.4 TTP-based Classification System Pipeline 56
3.5 Percentage of samples matching various MITRE ATT&CK TTPs across each dataset, with a logarithmic Y-axis. 59
3.6 RADAR's (left figures) and RB-RADAR's (right figures) performances when using the 3 aggregation policies P1 (top), P2 (middle) and P3 (bottom) while varying the user defined threshold. 65
3.7 ROC Curves comparing RADAR and RB-RADAR. 66
3.8 Network traffic graph views before (Figure (a)) and after (Figure (b)) the TTP Detection phase. The coloured nodes and edges represent hosts and flows respectively, which correspond to a detected TTP. In this example, the red flows represent the presence of *T1071 - Application Layer Protocol* while the blue flows represent *T1571 - Non-Standard Port*. 69

4.1 Visualisation of the 5 newly-created datasets. 86
4.2 Performance when varying the size of the training set. On the x-axis the percentage of randomly sampled datapoints from the training set ($\{0.1\%, 1\%, 10\%, 100\%\}$) can be found. 89
4.3 Performance when varying the size of the bins to create the BoF. On the x-axis the size of the bins can be found. 92
4.4 Performance when injecting different percentages of benign flows in the samples at testing time. On the x-axis the percentage of injected benign flow can be found. 94

5.1 Average and confidence intervals of F_1-score, Precision and Recall for T1105 while varying the threshold θ of the TTP label. 122
5.2 Performance of the models when varying the size of the training set. On the x-axis the percentage of randomly sampled datapoints from the training set can be found. 124

5.3 Performance of STTP2STTP and S2STTP when tested with varying percentages the noise ratio, defined as the ratio between the number of injected benign flows and the number of flows in the original sample. 129

List of Abbreviations

IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
HIDS	Host Intrusion Detection System
EDR	Endpoint Detection and Response
SIEM	Security Information and Event Management
SOC	Security Operations Centre
TTP	Tactics, Techniques, and Procedures
CTI	Cyber Threat Intelligence
OSINT	Open Source Intelligence
LotL	Living off the Land
PCAP	Packet Capture
NetFlow	Network Flow
YAF	Yet Another Flowmeter
RDB	Relational Database
C&C	Command and Control
MITM	Man-in-the-Middle
NTP	Network Time Protocol
TCP	Transmission Control Protocol
UDP	Universal Datagram Protocol
HTTP(S)	Hypertext Transfer Protocol (Secure)
IP	Internet Protocol
DNS	Domain Name System
SSL	Secure Sockets Layer
RTT	Round Trip Time
UID	Unique Identifier

ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
TPR	True Positive Rate
FPR	False Positive Rate
TNR	True Negative Rate
FNR	False Negative Rate
BoF	Bag of Flows
BoW	Bag of Words
AI	Artificial Intelligence
ML	Machine Learning
NN	Neural Network
DNN	Deep Neural Network
FFNN	Feed-Forward Neural Network
RNN	Recurrent Neural Network
DT	Decision Tree
AdaBoost . . .	Adaptive Boosting
LR	Logistic Regression
SVM	Support Vector Machine
RF	Random Forest
XGBoost . . .	EXtreme Gradient Boosting
ReLU	Reactivated Linear Unit
SNR	Sample Noise Ratio

Defeat is not declared when you fall down,
it is declared when you refuse to get up.

— Kristin Ashley, *The Will*

1 Introduction

Contents

1.1 Motivation 1
1.2 Scope of this Thesis 4
1.3 Ethical Considerations 4
1.4 Outline 5

1.1 Motivation

Malware detection is a notoriously complex and well-studied problem. Part of the challenge stems from the nature of the task itself—it is not static—malware evolves, adversaries learn and get smarter, technology advances and brings about previously unimaginable possibilities. Thus, for security researchers who have an interest in preventing or limiting its malicious capabilities, the problem of malware detection is more akin to an ever-evolving art. Much like the fabled Hydra [136] in Greek mythology, who would regrow two heads whenever one of its nine heads was cut off, "solving" a challenge within the domain of malware analysis is often just a temporary victory after which the adversaries in this digital arms race will only come back harder. This one of the reasons why the research literature for malware analysis is strewn with systems, methods, and algorithms that claim success at rates as high as 99% for metrics such as Accuracy, F_1-score, and AUC, and yet the problem of stopping malware still exists and shows no signs of going away. Thus, often the best strategy is one of continuous improvement and advancement, taking into account current malware capabilities and preparing for what may come next.

In this thesis, we aim to tackle the problem of malware detection keeping these principles in mind. The methodologies we propose should be useful not just in the context of malware today, but should help guide the development of future malware detection systems as well. Specifically, we tackle the problem of network-based malware detection, i.e., detecting malware that operates on, propagates using, or utilises computer networks for any purpose. This choice of problem is deliberate, since studying network-based malware has several advantages over other types of malware (such as host-based malware)—centralised observability, coverage and deployment of detection systems, simpler generation and collection of standardised data, to name a few—these, along with the general evolution of malware detection approaches are discussed in greater detail in Chapter 2.

Systems that are designed to tackle this problem are known as Network Intrusion Detection Systems (NIDS), and are an extensively researched part of the malware analysis literature. Early literature in the field relied on rule-based approaches wherein the rules were crafted by experts who knew definitive methods to spot malicious activity and identify malware. However with the passage of time and rapid advancements in the field of Machine Learning (ML) and Artificial Intelligence (AI), modern intrusion detection systems have broadly come to rely heavily on data-driven approaches. These approaches are effective because they leverage large volumes of data to train their ML/AI models, which are then used to spot anomalies in the data that would otherwise be non-trivial to identify using other approaches. As such we seek to utilise and learn from as much of this existing research as possible, when designing our approach. Our contribution to the field is by taking the best of both these worlds and creating a combined approach to tackle this problem—one that utilises data-driven learning-based approaches and exploits existing human domain knowledge, in a systematic and automated manner. We achieve this by making use of the knowledge about adversarial behaviour, that is collated by experts, and described within the MITRE ATT&CK framework. This extensive corpora of domain knowledge available today is then injected into ML-based NIDS that we design to exploit this information. The Tactics, Techniques, and Procedures (TTPs) described in the ATT&CK framework map the fundamental building blocks of all known malicious activity performed by malware. The fundamental intuition behind our approach is that this mapping of adversarial behaviour to TTPs can be used to model, identify, and classify malicious activity. Applying this to our problem domain, the key idea is to create and allow NIDS that can exploit this TTP information, alongside conventional network-based information, to better detect malware. Given that such domain knowledge is usually only within the purview of human operators, our hypothesis is that if utilised correctly, this information when combined with automated learning-based malware detection techniques can result in significant advantages for those defending against malware.

The goal of this thesis is to: (*i*) test this hypothesis and determine if such an approach is even feasible, (*ii*) determine the various methodologies by which this approach can be exploited, (*iii*) rigorously evaluate the performance of systems based on this approach against real-world malware, and (*iv*) establish a clear understanding of the benefits and trade-offs of such an approach when compared to existing approaches. Additionally, through the course of this thesis we will attempt to answer several broad questions regarding our proposed research, such as:

1. Can background knowledge in the form of TTPs be exploited for the purpose of malware detection?

2. Is the presence of TTPs alone sufficient for classifying a sample as malicious? If not, what level of TTP information is required to make a determination regarding maliciousness?

3. What are the limitations of existing systems that a TTP-based malware detection system would improve upon?

4. Can a TTP-based approach only be utilised by a specialised NIDS? Is it possible to create a system where any ML-based NIDS can exploit this TTP information in a "plug-and-play" manner?

5. Given the importance of TTPs in this process, is it possible to identify malware using specific TTPs? Could a combination of such precise TTP-based detection decisions be more useful in detecting malware, than a more generic TTP-based detection method?

1.2 Scope of this Thesis

The focus of this thesis is designing systems and methodologies to exploit expert domain knowledge in the form of TTPs from the MITRE ATT&CK framework, for the purpose of detecting malware using network-based data. We utilise methods from both static and dynamic analysis techniques of malware analysis, in addition to combining them with various ML-based approaches. The datasets we collect and curate are analysed in a dynamic analysis environment that extracts their network traffic. Contrary to most work in this field, we use on average 2-3 orders of magnitude more malware samples in our datasets, which are then used for analysis, evaluation, and producing statistically significant results.

As such, ontologies other than MITRE ATT&CK that describe adversarial behaviour are out of the scope of our analysis. Similarly, the focus of this thesis is evaluating this novel approach within the context of network-based malware, and we do not consider host-based malware. Lastly, the broad goal of this thesis is malware detection and thus we do not cover other aspects of malware analysis, such as malware family classification or malware type classification.

1.3 Ethical Considerations

While this research does deal with millions of samples of real-world malware, the design and goals of our experiments ensure that there is virtually no risk posed by them. The datasets we create are collated from publicly available sources, and the hashes corresponding to them are passed on to a dynamic analysis environment, which generates the raw network traffic captures that we want to use in our systems. Given the choice of our network data format (i.e., NetFlows), these packet captures are converted to a format that contains only metadata and no data payloads. As a result, no malware samples are ever directly executed by us, and by the time our systems are analysing the data the samples have generated, all potential malicious data payloads have been stripped from them. As such, there is very little risk of inadvertent infection or further propagation during the course of our research and experiments. Publicly releasing our datasets also poses no ethical concerns since our sources were already publicly-available. Furthermore, given that our datasets would only contain NetFlow data, there is no risk that the malware corresponding to that data can be somehow extracted and used by a malicious actor.

1.4 Outline

The rest of this thesis is structured in the following manner:

- **Chapter 2** places this thesis in the context of the related work in the field. It also introduces several concepts used throughout this thesis and provides crucial background needed to understand the rest of the chapters. We introduce the MITRE ATT&CK framework and describe research that has used TTPs within a cyber security context. Then we describe the advances in the field of malware detection and the various methods used for malware analysis. Lastly, we look at work beyond the scope of malware detection, specifically some of the research related to malware family classification.

- **Chapter 3** describes the design and evaluation of the first network-based malware detection system that utilises TTPs. We design this system, called RADAR, to be both extensible and explainable—useful properties when analysing constantly-evolving malware. We evaluate RADAR using a dataset of over 2.2 million samples, comparing its performance to other systems, quantifying the effect of our TTP-based Classification system, and comparing the results of using various policies for detecting malware.

- **Chapter 4** introduces a methodology by which TTP information can be exploited automatically, without any changes or *a priori* knowledge, by any ML-based NIDS. We design a methodology that converts network traffic corresponding to samples into Bags of Flows, and injects TTP information into any ML model by transforming them into model features. We evaluate this approach with 7-different types of ML models and while using 5 new datasets containing over 1.5 million samples. We further test the robustness of this approach under various challenging conditions, including when subjected to adversarial attack.

- **Chapter 5** proposes a novel methodology that models malware detection as a multi-label classification problem and generalises several detection approaches, including the one proposed in Chapter 4. This enables us to: (*i*) detect malware more precisely on the basis of individual TTPs, and (*ii*) identify the malicious usage of uncommon or rarely-used TTPs. This novel approach allows us to detect malware at both the sample-level and the TTP-level, and thus treat the overall task of malware detection as both a binary classification and multi-label classification problem. We evaluate this approach on 5 datasets containing over 1.5 million samples, and compare its performance to state-of-the-art systems in this domain. We also evaluate this system under various conditions such as when relying on limited training data, being subjected to camouflage attacks, or when varying the detection threshold of individual TTPs.

- **Chapter 6** summarises the results of this thesis and discusses avenues for future work in this domain.

I would rather have questions that can't be answered than answers that can't be questioned.

— Richard Feynman

2 Literature Review

Contents

2.1 Introduction **7**
2.2 The MITRE ATT&CK Framework **8**
2.2.1 Background 8
2.2.2 TTP Usage in Cyber Security Research 10
2.3 Malware Detection **12**
2.3.1 Content-based Approaches 12
2.3.2 Behaviour-based Approaches 19
2.3.3 Machine Learning based Approaches 29
2.3.4 Graph-based Approaches 36
2.4 Malware Classification **38**

2.1 Introduction

Malicious code is "any code added, changed, or removed from a software system to intentionally cause harm or subvert the system's intended function" [118]. Software comprising of malicious code is commonly referred to as malware, and it happens to be one of the most potent tools for information theft, fraud, abuse and compromise in the modern times [98]. That is what gives the field malware analysis its importance—it is a means of countering and neutralising this threat.

In this chapter we shall discuss various tools, techniques, and methods for malware analysis. First, we introduce the MITRE ATT&CK framework and provide some background about the Tactics, Techniques, and Procedures (TTPs) described within it. We then discuss how the ATT&CK framework and its TTPs have been utilised in existing literature within the context of cyber security, and lay the foundations for how they can be useful in tackling the problem of malware detection. The next section focuses on the general problem of malware detection, and traces how malware and the methods devised to counter it have evolved over the years. We broadly discuss the differences between content-based and behaviour-based approaches to malware detection, while also presenting the strengths and shortcomings associated with each approach. Our review of the existing literature covers everything from static analysis to dynamic analysis techniques, and explores methods utilising both host-based and network-based analysis techniques. We also highlight the prevalent challenges to this kind of research—obfuscation, polymorphic variants, encryption, and zero-days to name a few. We then take a deeper look at some of the modern methodologies employed in malware detection nowadays—such as ML-based and Graph-based malware detection systems. We employ some of these techniques in our research and hence these sections should provide a broader context to our choice of methods and experimental design.

Lastly, we briefly discuss research beyond the scope of malware detection, namely malware classification. Given that malware detection is simply one, albeit important, piece of the larger malware puzzle, we discuss why analysis beyond detection is needed in the first place, and the benefits conferred by classifying malware into their various malware families.

2.2 The MITRE ATT&CK Framework

2.2.1 Background

The MITRE ATT&CK® framework [126] is a globally accessible knowledge base of post-compromise adversarial tactics and techniques, based on real-world observations. It represents a behavioural model that is described by a wide-variety of adversary tactics, techniques, and procedures (TTPs). More specifically, within the context of ATT&CK:

- A *tactic* signifies a tactical goal of the adversary. It is the reason why the adversary performs any action.
- A *technique* represents the method by which an adversary achieves a tactical goal. A *sub-technique* within a technique corresponds to a more specific action within the given method.
- A *procedure* is a specific implementation of the method by which an adversary fulfils a tactical goal.

As an example, an adversary may dump OS credentials (*technique* T1003) to achieve credential access (*tactic*). This may be done by dumping the contents of `/etc/passwd` or `/etc/shadow` (*sub-technique* T1003.008), using a tool such as LaZagne[1] (*procedure*).

The MITRE ATT&CK framework is updated bi-annually to keep it up-to-date with the evolution of malware and novel adversarial approaches. The current version of the ATT&CK framework for Enterprise features 14 tactics, 201 techniques, and 424 sub-techniques. This thesis will focus specifically on the techniques (and encompassing tactics) that utilise the network to achieve their goals, which is a smaller subset of the overall techniques present in the ATT&CK framework.

This thesis uses the ATT&CK framework (and the TTPs it encompasses) since it is the most-widely recognised and used ontology for describing adversarial behaviour. However there exist other ontologies, such as Lockheed Martin's Cyber Kill Chain [116] or the Diamond Model [28], which describe the entities, steps, and actions involved in a cyber attack in their own way. The definition of TTPs as well, while broadly accepted to be as defined in the ATT&CK framework, is sometimes defined and used differently in various systems and security architectures. For our research however, throughout this thesis, we will stick to the conception of TTPs as

[1]https://github.com/AlessandroZ/LaZagne

defined above. Additionally, while TTP is an umbrella term used widely in industry, our usage of this term within the context of this thesis will focus mostly on the "technique" part of the acronym. The techniques (and sub-techniques) described in the ATT&CK framework are the most important and relevant descriptors for the purposes of malware detection in our research. Thus, the usage of the term TTP should be understood to be synonymous with the technique in the given TTP, more so than the tactic or procedure, unless specified otherwise.

2.2.2 TTP Usage in Cyber Security Research

It can be argued that TTPs can be useful in a wide-variety of contexts, especially when it comes to identifying and categorising the activities of malicious adversaries [48]. This is due to the fact that:

1. The technology operated by an adversary constrains the number and types of techniques they can utilise to achieve their goals post-compromise.
2. There are a relatively limited number of such techniques, which themselves have to be executed on a victim's systems and are therefore further restricted by their technology stack, *e.g.* operating system.
3. Consequently, an adversary has to either rely on these techniques—which can be detected at low cost—or engage in the much more costly process of developing novel techniques.

However, in practice, TTPs are mostly utilised to understand and analyse malware behaviour. In particular, TTPs from the MITRE ATT&CK framework have been used extensively to study the rapid growth of novel malware and malware families, and better understand the kinds of threats they pose to us. For instance, Chierzi et al. [38] take advantage of the MITRE ATT&CK framework to create an analysis methodology based on TTPs to track the evolution of Linux-based IoT malware. Al-Shaer et al. [9] exploit the ATT&CK framework to discover statistically significant technique inter-dependencies in a TTP chain using a hierarchical clustering approach that can help an analyst reason about adversarial behaviour and predict unobserved techniques. Oosthoek et al. [137] extract a number of different MITRE ATT&CK techniques from a dataset of Windows malware to provide an overview of established techniques and describe emerging trends in them. Fairbanks et al. [57] focus on identifying which part of the malware corresponds to a specific TTP. Their system automates the process of locating a TTP in a sub-part of the control flow graph that describes the execution flow of an Android malware executable.

Another method by which some systems have addressed understanding the evolving threat of malware is by targeting a key challenge in the domain of Cyber Threat Intelligence (CTI), namely parsing through large quantities of CTI reports to extract useful structured information (such as TTPs) that can be used by analysts to enable timely and cost-effective cyber defence from malware. Mendsaikhan et al. [121] tackle the challenge of parsing the ever-increasing set of vulnerabilities to generalise the threat landscape and make it easier for analysts to use them. They create a method to automate the mapping of vulnerability description to adversarial technique using a multi-label classification approach. Similarly, Legoy et al. [107] evaluate and build a tool, rcATT, that uses multi-label text classification model to automatically extracts TTPs from the unstructured text of CTI reports. Husari et al. [84] build a similar tool, TTPDrill, that automates the analysis of CTI reports and maps their unstructured text to MITRE ATT&CK TTPs using natural language processing and information retrieval techniques.

Some other works use MITRE ATT&CK techniques as part of a larger system pipeline to help with specific detection tasks. For instance, Kuppa et al. [104] map process execution logs to MITRE ATT&CK TTPs and use them as features to build a group anomaly detection model that uses adversarial autoencoders. Hassan et al. [79] on the other hand, extract ATT&CK TTPs and use them in tactical provenance graphs to create a more effective Endpoint Detection and Response (EDR) tool for detecting APT attack campaigns.

However, to the best our knowledge, there exists no prior research that directly uses TTPs for the purpose of malware detection. Through the course of the rest of the thesis, we shall describe exactly how such systems can be designed and built, while providing their own unique advantages and performance benefits.

2.3 Malware Detection

The problem of malware detection can be broadly divided into two categories based on the method of malware analysis: content-based and behaviour-based. However, there is no clear taxonomy by which all possible malware analysis methodologies can be neatly separated. Therefore, we choose these two broad categories and delve into other types of methodologies while exploring them. Secondly, we further pick two relatively modern methodologies that are popular in malware detection systems today—ML-based and Graph-based approaches—and explore them in greater detail. These malware analysis techniques allow analysts to determine the nature of a threat—what are the risks associated with this malware, what are its intentions and capabilities—and deal with it in the appropriate manner. The process of analysing malware also generates insights into the operational tactics of a malicious actor, such as the current attack vectors being utilised, the TTPs used for exploitation and propagation, and the potential targets of an attack. This is valuable information that can be used to identify both existing and novel malware, and enable the creation of defensive measures against future threats. In this section, we shall explore the various malware analysis methods in greater detail.

2.3.1 Content-based Approaches

2.3.1.1 Static-feature-based

The process of analysing malicious software without executing it is known as static analysis. There are several methods by which features can be selected in order to identify detection patterns in malware, such as—string signatures, byte-sequence n-grams, syntactic library calls, control-flow graphs and opcode frequency distribution [67]. Static analysis can be applied to both the malicious code within an executable or directly on the binary representation itself. Each approach has its drawbacks—for instance, the source code of malware is not always available to analyse, whereas analysing the binary directly leads to a loss of information such as the size of data structures and variables [56]. Some of these methods require the malicious executable to be unpacked, disassembled or dumped (from memory) in order to be analysed. For instance, tools such as IDA Pro [4] and Ghidra [3] can be used to disassemble binary files into assembly-level code that reveals the programming logic and structure of the binary executable file. Other tools such as LordPE [6] and OllyDumpEx [7] can be used to dump protected code in a system's memory to a file, in the case of packed executables that are hard to disassemble.

One of the earliest attempts to utilise such an approach was by Lo et al. [111], where they attempted to identify *telltale signs* (such as duplicated system calls, isolated/independent code, anomalous file accesses) to filter programs and classify them as computer viruses based on the telltale signs they exhibited. Their approach could identify similarly structured but previously unseen malicious code. To handle malicious code that was not detected by the MCF (Malicious Code Filter), they suggest updating the telltale signs used by the MCF to handle novel malicious code.

While not a focus of our research, another early related area of malware research was based on using formal methods during static analysis. Research in this area was based around the paradigm of model checking, and using those models to verify security properties and identify vulnerabilities [34, 35, 60]. For instance, these methods have been used to identify memory corruption flaws or verifying the correctness of models for various UNIX applications.

Signature-based detection techniques like MCF, while useful, can be evaded if the signature of a malware can be sufficiently changed—a property malware authors came to adopt rapidly. In response to these limitations, several new detection methods were proposed that used "higher-order" properties of the malware to describe it. The idea behind this approach being that these properties could capture intrinsic characteristics of the malware, and thus make it much harder for malware to evade detection by modifying itself [97]. One such technique was using n-gram analysis. An n-gram is defined [114] as a "contiguous sequence of n items from a given sample of text or speech", and in the context of malware analysis usually refers to a sequence of n bytes in the binary representation of a program. Proposed by Li et al. [108, 109] this method uses n-gram analysis to conduct a static inspection of the statistical byte sequences of binary content. The intuition being that n-gram distributions of malicious files would be significantly different from benign files, and this difference in portions of contiguous byte sequences could effectively identify malware. An additional benefit of this approach was that it could be used to identify malicious "files" and not just executables since it analysed the n-grams in the binary content of the file, i.e., it could detect embedded malware in common files or malware posing as a document. They achieve this by using statistical methods on a labelled training dataset of documents and characterising both normal and malicious document content. Their experimental analysis was limited to Word documents, but their results showed that a 5-gram was a sufficient length to capture enough information about the binary file content for the dataset used in their experiment. Crucially, their statistical n-gram analysis yielded better results in terms of malware detection than commercial off-the-shelf anti-virus (AV) scanners.

Another such higher-order property was the Control Flow Graph (CFG) of a program—a graphical representation of all possible execution paths—which became the basis of several malware detection techniques [101, 27]. The intuitive benefit of this approach being that it could be used to detect all polymorphic variants of a malware on the basis of their shared CFG structure [97]. One such notable example of this method was proposed by Christodorescu et al. [39] who used CFGs to develop a methodology for detecting malicious patterns in executables. Their approach was different from existing techniques at the time because they viewed the problem of malicious code detection as an obfuscation-deobfuscation game. This essentially led them to tackle the issue of obfuscation that malware authors were adopting in response to existing static detection techniques—their problem statement thus became one of finding if instructions in obfuscated software were semantically equivalent to known malicious instructions. Secondly, unlike existing static analysis techniques which analysed only source-code their framework could analyse executables. This is particularly useful because source-code is not always available during malware analysis. Their framework works by creating abstract representations, or "generalisable" forms, of both the executable being tested and the known malicious code. The creation of this generalisable form allows them to account for common obfuscation techniques and counter them accordingly in this representation. They then transform the representation of the executable into another abstract representation (in this case, a Control Flow Graph), one that represents the code that created the executable. This transformation is made possible by IDA Pro [4], an interactive disassembler, and CodeSurfer (now CodeSonar [2]), a program-understanding tool that can take as input CFGs or call graphs and perform various static analyses on them. This CFG is annotated further such that patterns within it can be compared to the patterns present in the generalisable forms of known malicious code. The annotated CFGs from a malware are then compared to generalisable forms of known malicious code using static analysis techniques.

These techniques by no means cover all the various sophisticated static analysis methods that can be used to capture and detect malicious functionality. Notable contributions to the research literature include approaches that use symbolic execution [100], model checking [94], or techniques from compiler verification [41] to find implementations of specific functionality in arbitrary code fragments. The key idea in all these approaches being that certain functionality of these malicious programs can always be identified regardless of the specific instructions that are used—which is what makes these techniques useful and effective [97].

Despite their versatility and effectiveness in detecting malware, static-feature-based analysis techniques have a number of drawbacks, which we shall discuss further in detail, that make them unsuitable as a sole malware detection method. These drawbacks, observed and studied since the very beginning of malware detection research, include:

1. The availability of source code for malware can pose a problem, since often it is only the binary executable that is available and not its underlying code [56].

2. The analysis of binary files has its own set of challenges. For instance, the results of static analysis can be ambiguous if the malware uses self-modifying code techniques, such as packers. Similarly these techniques cannot analyse parameters that are not statically determined, such as the current date/time or indirect jump instructions [56]. Additionally, analysing binaries can be challenging if their malicious payload is encrypted or obfuscated; and even if it is somehow accessed and decompiled, there is no guarantee of getting the same instructions as the original malware.

3. Most importantly, once the techniques used in static analysis are known, malware begins to adapt specifically to evade these techniques. Naturally, this is not ideal from a malware detection perspective. Additionally, these evasion techniques can be quite sophisticated and hard to overcome using existing methods. Some of these evasion techniques include the following:

   (a) Using opaque constants to thwart static analysis. These are primitives using which arbitrary constants can be loaded into a register in a manner that is opaque to analysis tools, i.e., they cannot determine the value in the register. Using this technique as the building block, Moser et al. [128] were able to build obfuscation transformations that were capable of obscuring program control flow, disguising access to local and global variables, and interrupting the tracking of values held in processor registers.

   (b) Using polymorphic malware that can morph itself in order to evade detection. A common method of "morphing" is to encrypt the malicious payload of the malware and only decrypt it during execution. The methods of obfuscating the decryption routine itself can be varied and complex, including transformations such as: nop-insertions, code transposition (changing the absolute order of instructions while maintaining semantic integrity using additional jump instructions), and register reassignment (permuting the allocation of registers) [40].

(c) Making use of metamorphic malware, that uses more complex obfuscation techniques to evade malware heuristic detection. Much like polymorphic malware, when such malware "morphs" or replicates, it modifies its code in various ways: code transposition, equivalent instruction sequence substitution, change of conditional jumps, and register reassignment [173]. But unlike polymorphic malware, they can go a step further and "weave" themselves into the code of another program. This makes them nearly undetectable to heuristic-based approaches since the new malware has a mixed codebase and the malicious portion is not necessarily at the beginning of the program, thus obfuscating their entry-point [40].

(d) By executing malware solely in memory. This type of malware resides and executes its entire payload from the main memory of a system. Since it does not access the file system it is also known as fileless malware [141, 103]. Malware of this kind is particularly insidious since it leaves no traces on the filesystem, and hence for instance if it executes its payload and then deletes itself, cannot be analysed using static approaches.

(e) Using obfuscation techniques to evade detection. There are a wide range of obfuscation techniques that a malware can apply to evade detection [40], including techniques such as polymorphism and metamorphism. The key function in each obfuscation technique is to maintain its functionality while changing its appearance [97]. This technique as a result is rather effective in thwarting analysis techniques that rely on static pre-defined signatures for detection.

4. Most advanced static-analysis techniques are too slow to be used as a real-time system. The trade-off of using more comprehensive analysis techniques is the run-time of the detection algorithm. For instance, one such approach that utilised malware semantics for detection took in the order of minutes to

complete the analysis [41]. Such techniques are thus not suited for real-time detection systems [97].

There is a constant arms-race between security researchers to develop better techniques to detect malware and malware authors to write more sophisticated malware that can evade detection. While some static analysis techniques have been better designed to handle evasion by obfuscation (such as [39] which can handle polymorphic variants of the same malware), others have been designed to tackle this problem directly, i.e., deobfuscating malware. Significant research contributions have been made in the area of counter-evasion techniques that can be used to build generic and automated systems that can unpack malware. PolyUnpack [156] for instance, was the first such automated unpacking technique to be used against obfuscated malware. This system first builds a static model of the program using static analysis. Then it executes the program in a sterile, isolated environment and uses the Windows debugging API to track every step of the process execution, and compares the output to the existing static model. As soon as it detects instructions that are not present in the static model, it saves those new instructions and considers the program unpacked. Renovo [90] is another such system that is similar to PolyUnpack since it too uses a fine-grained execution monitoring approach and it also detects an unpacked program when new and unknown code is executed. It differs from PolyUnpack as it resides outside the program's execution environment and supports multiple layers of unpacking. OmniUnpack [115] and Eureka [161] are two more such frameworks that take a slightly different approach to the problem. They too track program execution, waiting for the static model to get to a stable state in order to determine that the program is unpacked. Where they differ from previous approaches is that they use coarse-grained tracking instead of fine-grained tracking. OmniUnpack tracks execution at the page-level while Eureka tracks execution at the system level [161]. In the experiments run with these various techniques they were able to detect the vast majority of packed malware, thus indicating the relative success against the ever-evolving malware industry.

Nonetheless, as this literature makes clear, while there are advantages to using static analysis and there are techniques to handle some obfuscation methods being deployed by malware, there still remain significant drawbacks to this approach. Evasion techniques in particular can be rather effective against static approaches. Hence, it is unlikely that static-feature-based analysis alone is sufficient for effective malware detection—there is a need to develop analysis techniques that are resilient to modifications such as the ones discussed above and are still able to reliably analyse malicious software [56].

2.3.2 Behaviour-based Approaches

Behaviour-based approaches on the other hand detect malware by analysing the execution of a program and the effects it has on the environment being observed. These techniques are largely classified as dynamic analysis in research literature. Depending on the specific environment being observed, these approaches can be divided into two separate categories: host-based detection and network-based detection. Host-based detection analyses the behaviour of program execution at the OS-level and measures its effect on the machine it is being executed on itself. Network-based detection on the other hand analyses data (i.e., network traffic) at the the network level and bases its malware detection decisions on their observed network properties instead.

Dynamic analysis techniques offer several important advantages over static analysis techniques. Firstly since the analysis is done at run-time, the malware unpacks itself and begins executing its malicious payload, thus completely doing away with the problem of packing and obfuscated code interfering with malware analysis. The second major advantage being that this type of analysis can be automated and hence can be conducted on a much larger scale. At the same time, they do also have some drawbacks, especially when contrasted to static analysis approaches. In fact it is broadly agreed upon in the research community that some combination of static and dynamic malware analysis techniques would be more effective than either technique individually [109, 42]. Nonetheless dynamic analysis provides a versatile and effective means of malware detection, while being able to overcome some of these challenges. We shall discuss these techniques and their capabilities in more detail in the subsequent section—in the meantime it is useful to be aware of some of their inherent common weaknesses to begin with. Some of these drawbacks include:

1. They expose the underlying system on which they are stored to risk [111]. This is less of a concern today due to the prevalence of virtual testing environments (such as Virtual Machines), but this by no means is a guaranteed solution to the problem. Malware has become sophisticated enough to detect when it is being executed in a virtual environment [37, 202] and can simply choose to not exhibit any malicious activity while it is in such an environment.

2. These techniques can only identify and thwart malware based on its activities, but are unable to identify malicious code if it is dormant [111]. This for instance can be caused by a logic bomb in a malware, i.e., program logic that instructs the malware to execute its malicious payload only at a specific date and time.

3. Dynamic analysis techniques rely heavily upon the execution traces of malware samples to identify malicious behaviour. But these traces often represent a single execution cycle of the malware, which may not exhibit the full functionality of the program. This incomplete code coverage can lead to certain program behaviours not being analysed that might be crucial in detecting malicious activity.

4. When analysis techniques that operate at run-time detect malware, they resolve the problem by either halting execution of the program completely or by requiring human-intervention. Thus, these techniques are unsuitable for systems that are critical or have high availability requirements, and operate without constant human supervision [111].

2.3.2.1 Host-based Analysis

As the name suggests, host-based behavioural (or dynamic) analysis techniques rely on observing the behaviour of malware on a host machine, and making inferences using the information learned from analysis at this level of abstraction. This usually involves executing a malware sample within a controlled environment and monitoring its actions in order to detect malicious behaviour [66]. Several techniques can be utilised for the purpose of dynamic malware analysis [56], such as function call monitoring [42, 197], function parameter analysis, information flow tracking [55, 130, 129, 134], instruction traces and autostart extensibility points [190].

One such technique that uses function call monitoring was proposed by Christodorescu et al. [42]—they proposed building an automated system that could derive a specification of malicious behaviour (or *malspec*) exhibited by a given malware. By comparing the specifications of known malware with benign software, their system could identify programs with malicious behaviour. By using a high-level abstraction of the malware's behavioural properties instead of specific characteristics of the malware (such as instruction sequences), this detection method is more robust in the face of evasion techniques that change the form of the malware but preserve its behaviour, such as obfuscation and packing. While other systems utilising similar high-level abstractions to describe malware needed those specifications to be created manually—a slow, laborious task that required expert knowledge—their system could create these malspecs in an automated manner based on the given malware samples. The key to such an approach is creating a specification that accurately captures malicious behaviour and is not present in benign software. To this end, their algorithm uses execution traces to create system-call graphs for each malware, and then derives a specification based on the minimal differences between the system-call graphs of the given malware sample and the benign programs. Kolbitsch et al. [97] also proposed a similar approach that entails analysing the malware in a controlled environment and creating graphs characterising its behaviour. They too use system-calls for creating these behaviour graphs, but unlike malspec their framework monitors not only the system-calls but also their parameters and the arbitrary data flows dependencies between them. This allows them to create more granular and detailed behaviour graphs, resulting in higher accuracy in malware detection and a lower false-positive rate. They extract program slices that are responsible for information flows between the system-calls essential to the malware's behaviour. These slices are then compared to models created by the run-time behaviour of unknown programs to detect if their is a match, i.e., the program exhibits some recognised malicious behaviour.

Another dynamic analysis technique that falls under the ambit of information flow tracking is taint analysis. Tainting, or the process of tracking dynamic data flows can be quite useful in identifying malicious behaviour. For instance, one of the major drawbacks of dynamic analysis techniques is that they use single execution traces of a malware to reason about their behaviour, leading to the problem of incomplete code coverage and potentially missing important behaviour that was not exhibited. Moser et al. [129] tackle this problem by proposing a dynamic analysis technique based on tainting of the malicious code's inputs. They dynamically track some of the inputs to the program (such as current date/time, contents of a file, or results of an internet connectivity check) and identify the points at which these inputs are used to make control flow decisions for the program. They use this information to alter these inputs and observe the outcome of the conditional logic, thereby exploring additional, possibly useful, execution paths. This allows them to have a more complete view of a program's actions and thus improve malware detection. More recently, Küchler et al. [102] investigated this problem and attempted to determine the ideal time to observe a malware sample in a dynamic analysis environment. Their experiments tried to balance the competing requirements of time (observing a sample for as long as possible, so that no potential malicious behaviour is missed or left unobserved) and quantity (the number of samples that can be processed in a dynamic analysis environment within a given period of time, inversely proportional to the observation time, and thus affecting the size of the malware dataset studied and the statistical significance of its results). Their results indicated that most malware samples executed for less than two minutes or more than ten, and that the majority of a sample's behaviour could be observed with an observation threshold of two minutes. This period of execution was also found to be the most representative from the perspective of an ML classifier. This finding about the "ideal" observation time confirmed previous such results by Willems et al. [192], and played a major part in our selection of such parameters when conducting our own experiments with malware in dynamic analysis environments.

Taint analysis based techniques are not limited to this scope necessarily. For instance, Kirda et al. [96] create a malware detector focused specifically on identifying browser-based spyware. They focus on malware that exploits Internet Explorer's hooks to monitor user actions—this is done by exploiting the Browser Helper Object (BHO) interface. Their tainting scheme feeds browser events into BHOs and observes how they react to these browser events. Egele et al. [55] extend this scheme to track sensitive information flows processed by a browser and any BHOs in order to characterise the maliciousness of an unknown browser component. By performing this taint analysis in a modified Qemu environment (a system emulator), they are able to see if a BHO collects sensitive data and writes it to a disk or send it over a network, and identify this behaviour as suspicious. Yin et al. [201] take this method one step further by implementing *whole-system fine-grained taint tracking.* Their system, Panorama, works by loading a malware into their analysis environment, testing it automatically by introducing sensitive information into the system that is not meant for the malware, then monitoring the information access and processing behaviour of the malware sample, and finally analysing this information to detect malicious behaviour. To enable this kind of tracking of information flows they combine taint propagation at the hardware-level and the OS-level, and create *taint graphs* that represent these flows. Policies can then be created on the basis of these taint graphs to specify the characteristic behaviour of various malware and identify similar behaviour in an unknown sample. The natural downside of such a fine-grained system-wide tainting approach is the performance of such a system when compared to other approaches—a slowdown by a factor of 20. Hence while such a system is more likely to more accurately characterise malware behaviour, the performance cost may be prohibitive for some applications.

2.3.2.2 Network-based Analysis

Network-based malware analysis is another broad category of techniques used for detecting malicious activity originating from software programs. Unlike the static and dynamic analysis techniques discussed so far, which are both based on analysis on the end-host directly, these techniques instead focus on analysing malware at the network level. This for instance could imply analysing malicious activity based on the content or metadata of network traffic, instead of analysing the system-calls and control-flow of a program. This network-based approach has its own advantages when compared to other detection methods. These include:

1. A single sensor can be deployed in a network to monitor the traffic of all machines in that network [97]. This is a major advantage of this approach, especially compared to host-based approaches, since traffic can be analysed at a single point and the detection methods can be used to safeguard all the machines in the network. While host-based techniques generally provide a more detailed and concentrated analysis of malware, they also create the additional overhead of deploying the system on each host and then monitoring them [11]. Thus, instead of relying on securing infrastructure one machine at a time, this approach enables defences against malware to be scaled-up massively.

2. Network-based malware detection systems are also impervious to commonly-used evasion tactics such as obfuscation, polymorphism, metamorphism, and packing. Since all of these evasion techniques focus on changing the binary structure of a malicious executable while preserving its behaviour, they have no effect on a system that does not analyse the executable or its structure directly. This does not imply that similar techniques cannot be implemented to modify and cloak network traffic instead to enable evasion, just that the

most common forms of evasion techniques used by malware are rendered ineffective when conducting network-based analysis.

At the same time, such an approach is not without its drawbacks. Similar to other malware analysis techniques, network-based detection is best utilised knowing both its weaknesses and advantages. Some of these drawbacks include:

1. It cannot detect malware that does not generate network traffic. While such a feature is relatively uncommon in modern malware, (which generally use some form of network traffic for communication, exploitation, or propagation) it is nonetheless possible for a malware to infect a machine, execute its payload, and propagate further without using a computer network. Such malware are for all intents and purposes, invisible to a network-based malware detection system, and hence should be deployed in situations where the threat model does not include such malware or accommodates them with other mechanisms.

2. Network-based analysis approaches require additional action to be taken after detecting malicious activity. Since this method works at the network-level, it cannot identify the malicious program responsible or halt its execution. Depending on the granularity of the network traffic data it is using, it may identify the network location (such as the IP address) of an infected host, but requires human-intervention to perform the next steps of zeroing in on the malware and taking actions to prevent its execution.

Given the exponential growth of internet-connected devices and increasingly large-scale hyper-connected systems, these Network Intrusion Detection Systems (NIDS) have become a popular choice for ensuring security, confidentiality, and integrity of these systems. But these present conditions also create several unique challenges for such detection systems, these include—large traffic volumes, highly imbalanced data distributions, and separating anomalous network behaviour from "normal" network behaviour [11]. These systems also need to be able to operate effectively and adapt to changing network behaviour in an automated manner, without requiring constant manual tuning [179]. Thus, solving these challenges is critical for a network-based malware detection system that aims to be robust and reliable, even in the face of changing network traffic and unknown malware [172].

These NIDS (a network-specific implementation of general Intrusion Detection Systems [50]), thus monitor all network traffic above and including Layer 3 of the TCP/IP model (i.e., network, transport, and application layers) and analyse them to detect malicious activity. They can broadly be divided into 2 categories: anomaly-based and signature-based. Anomaly-based IDS operate by learning and defining a "normal" profile of the network traffic, and detect malicious or suspicious activity by identifying network traffic that deviates from this profile [68]. Some approaches exploit the similarity of network communications in infected hosts to detect malicious activity [76, 199, 77], while others use statistically anomalous properties of network traffic to analyse malicious behaviour [186, 187]. Signature-based IDS on the other hand detect malicious activity by monitoring the network for content-based signatures that are characteristic of known malware families [97]. These signatures can either be defined manually [75, 143, 154] or derived automatically by analysing a collection of malicious payloads [135, 167, 185, 93]. Similar to other host-based signature detection techniques, signature-based NIDS are quite effective at identifying known malware with a low false-positive rate. This same property however prevents such an approach from being effective in detecting new malware or variants of existing malware. To counter these challenges, the signatures used by these NIDS require frequent updates and the creation of new rules or policies. Anomaly-based detection techniques on the other hand are better suited to detect new or unknown variants of malware. An additional challenge for an attacker wishing to evade such a system is that they need to know what the "normal" profile of network activity looks like—something that is unique to every network [140]. Some of the drawbacks of such techniques are that they have a high false-positive rate, they are hard to scale to gigabit speeds, and attribution of an anomalous activity to a specific event is hard [140]. The high-volume of false alerts can be specially problematic since they require a human analyst to sort through and address them, resulting in actual malware remaining active for longer in the network while this time-consuming process takes place.

Anomaly-based NIDS

An example of one such anomaly-based IDS is PAYL which was proposed by Wang and Stolfo [186]. PAYL, or a payload-based anomaly detector, functions by analysing network payloads and learning a model for what normal payloads for a specific service in that network should look like. This is done by creating a *byte frequency distribution* of each payload, by modelling these payloads as n-grams. This distribution created by the n-gram analysis creates a model for “normal” payloads. In the anomaly detection phase, PAYL compares the distribution of an incoming network payload with the normal (or *centroid*) model on the basis of its Mahalanobis distance—a standard distance metric used to compare two statistical distributions. Any new test payload with a distance higher than a certain threshold is considered to be anomalous, and an alert is generated in response. The flagged network connection can then be filtered, rerouted or otherwise prevented from transmitting its payload on the given service, depending on the security policy of the network. This system was able to detect nearly 60% of the attacks it was tested against, with a key takeaway being an overall false-positive rate of less than 1%.

PAYL was the first system to demonstrate the effectiveness of an anomaly-based IDS to counter malware. In particular, it combined the benefit of a low false-positive rate as found in signature-based IDS, with the property of being resistant to common evasion tactics that plagued host-based analysis systems. However, this latter property was put to the test by Fogla et al.[64] who introduced a new class of malware evasion techniques called *polymorphic blending attacks.* These were a form of polymorphic attacks that could evade the byte frequency distribution model used in systems like PAYL. They leverage the simplicity of the statistical measures used in the model (a choice that allows the IDS to be high speed) to craft malicious payloads in a manner that will make them match the characteristics of the normal model. This is done by carefully choosing the characters of the encrypted malicious payload, as well as the characters used to pad the payload, such that the overall byte frequency distribution matches that of a payload considered normal by PAYL. They find that as long as they are able to observe the “normal” network traffic behaviour of a given network for some time, they can approximate the normal profile of certain payloads as measured by the IDS, and generate a polymorphic variant of a malware that has a similar profile and hence will successfully evade the IDS.

In a manner that is quite representative of the general “arms-race” that pervades the world of malware research, the authors of PAYL then created a new anomaly-based IDS known as Anagram [187], which is resistant to the kind of polymorphic blending attacks discussed above. They use bloom filters to store n-gram frequencies of both malicious and benign packets, and create a normal model using a mixture of different-sized n-grams. They also utilise *randomised n-gram modelling* in order to counter current and future mimicry attacks such as polymorphic blending. In addition they propose a feedback loop between their sensor and detectors to constantly improve their model’s accuracy and make it harder to evade. The use of Bloom filters makes this approach more memory and computationally efficient, while also being easier to train. Hence, the overall system outperforms the existing PAYL IDS in both detection and false-positive rate while also being resistant to the kind of polymorphic blending attacks that were successfully deployed against PAYL. This example demonstrates well the perpetual state of one-upmanship that exists between malware authors and the malware detection community.

Signature-based NIDS

BotHunter [75] is a good example of a signature-based network intrusion detection system. Unlike traditional signature-based detection systems like Bro [143] or Snort [154], that rely on specific signatures or policies to detect malware, they use the rule-based matching of these systems while also attempting to identify malware based on their behavioural properties. It is designed specifically to detect bots and botnet communication with their C&C infrastructure. This system models bot infections as a set of loosely ordered communication flows (known as *dialog* *correlation*), and looks for a threshold combination of sequences in order to consider it a bot. In addition, it uses Snort for its statistical anomalous payload detection and signature-based malware detection capabilities on both incoming and outgoing traffic. When BotHunter detects a sufficient sequence of alerts that match its infection dialog model, its correlation engine produces a consolidated report that captures all the relevant events and event participants in the infection cycle, which can then be reviewed by a human analyst. This system, while being effective, had its own limitations, such as not being able to analyse encrypted traffic and needing multiple bot infections in a network to detect bot activity [184]. Further research also improved upon on other aspects of this system. For instance, Ashfaq et al.[14] utilised the infection dialog model of BotHunter and extended it to incorporate the use of passive propagation mechanisms such as spam.

2.3.3 Machine Learning based Approaches

Advances in the field of Machine Learning, and the nature of the problem in malware detection and classification, have made Machine Learning (ML) based techniques a good fit for malware analysis. These techniques can assist in extracting knowledge about the properties of various malware, help reason about their maliciousness, and make the entire process more adaptive and automated. Two of the key challenges in malware intrusion detection research are the high rates of false-positives and false-negatives, and the problem of adapting these systems to constantly evolving malware behaviour [194]. ML-based NIDS have specifically been created to tackle these challenges by improving the accuracy of these systems and providing a framework that can account for the evolution and variations found in malware.

Designing such models that are capable of learning from data is by no means a trivial process, especially in the context of intrusion detection systems. Several challenges to ML-based techniques have been documented in the research literature [168, 194]. Some of these include:

1. The high cost of errors. Since these systems are designed to learn continuously and update their models based on past beliefs, inaccurately classifying a

sample as malicious or assigning it to the incorrect family, can propagate errors much further and adversely impact the reliability of the model if not spotted and rectified early.

2. Variability of benign traffic. While malicious samples of data or traffic are easier to define because of their clear purpose, benign traffic is a much broader and harder-to-define category. This can affect how some models learn to classify what really constitutes a benign sample, since there is no clear definition of such behaviour.

3. Challenges in performing sound evaluations. In most cases the samples being tested do not have labels or ground-truth that can be used to definitively measure the accuracy of a model, and hence labels are assigned using other techniques (such as by using signature-based methods such as Bro [143], Snort [154] or VirusTotal [178]). The accuracy of this label assignment directly impacts the reliability with which the accuracy of a model can be evaluated.

4. Operation in adversarial settings. A major drawback of learning based techniques is that they are susceptible to other learning based techniques designed specifically to defeat a given model, i.e., adversarial ML. These risks are also a concern in the context of IDS since malware could be designed in a way that is optimised to evade such systems.

5. Huge network traffic volumes. From the view of a purely machine learning perspective, more data is always preferable in order to train models more reliably. But when deployed in an IDS, these models need to be able to handle large amounts of data while maintaining system speed and performance, such that they can be used in a real-world setting.

6. Imbalanced attack class distribution. In contrast to the over-generalisability of benign data, malicious data usually constitutes a small portion of the overall network traffic. Hence, learning from a small corpus of malicious samples may

not let the model effectively capture the inherent properties and behaviours of malware.

7. Lack of reliable real-world datasets. Many ML-based IDS rely on old datasets to train their models, such as the KDD CUP 1999 [5] or the DARPA Intrusion Detection 1999 [1] datasets. Given the rapid pace at which malware evolves, new malware emerges, and novel attacks techniques are utilised by it, these datasets are no longer sufficient to train or evaluate modern systems. However obtaining new datasets is not a trivial exercise since network traffic contains confidential and sensitive data, which an organisation may not want to share. Even if such data is shared, there are usually restrictions on its publication to preserve privacy. Lastly, datasets of such high volume are also often unlabelled and lack ground truth data, which makes system evaluation much harder.

Network-based intrusion detection systems that utilise machine learning for malware analysis, have used a wide variety of algorithmic approaches—these include Decision Trees [99, 170, 132], Naive Bayes [139, 12, 58], Rule Induction [154, 85], Neural Networks [157, 29, 200], Support Vector Machines [63, 82], Genetic Algorithms [81, 91], Fuzzy Logic [174], Data Mining [106], Random Forests [59, 203].

These ML-based techniques can be used in a multitude of ways to improve intrusion detection over a network. Yin et al. [200] propose one such system that uses Recurrent Neural Networks (RNNs) for network intrusion detection. Their system is able to achieve high accuracy in both binary (separating malicious traffic from benign traffic) and multi-class classification settings (identifying types of malicious attack traffic, along with benign traffic). They compare their approach with traditional classification approaches (such as J48, Naive Bayes, Random Forests) and find that their system has a higher accuracy and detection rate, with a low false-positive rate. Hence their model is capable of both identifying an intrusion and recognising the type of intrusion more accurately.

Zheng et al. [205] on the other hand tackle another challenge—accurately identifying benign network traffic and behaviour. As discussed above, the variability of benign traffic is a hard problem for ML-based NIDS. For instance, a sample not marked as malicious does not necessarily imply it is benign—it may contain untrained malicious network traffic that the classifier has not learned to identify yet. Proceeding by assuming otherwise leads to an intrusion detection system with a high false-negative rate. The authors propose a system that leverages differences between the properties of benign and malicious traffic (such as the amount of inbound/outbound traffic, connected devices, and communication frequency) to extract features from TLS communication channels. These features are then used to train a clustering model based on the DBSCAN algorithm, that can accurately identify benign traffic. They find that their method outperforms other approaches in terms of processing efficiency, and is much more reliable for mining benign TLS channels.

Shibahara et al. [165] use ML-based techniques to determine whether dynamic analysis of a sample should be suspended on the basis of network behaviour. Typically a malware sample is analysed in an IDS for a short period of time to identify key behavioural or structural properties that can be used for malware detection. The short duration though does not guarantee that the sample will showcase all of its malicious behaviour in that period. Conversely, every sample cannot be analysed for a long duration if an IDS has to be effective in a real-world setting. Hence the authors propose a method to increase the efficiency of dynamic analysis by suspending the analysis when the malware stops its activities. They do this by leveraging two common characteristics of malware communication—change in communication purpose and the common latent function. To this end, they focus on the similarity of data structure between malware communications and natural language. They use an RNN to model these features and obtain a decision as to whether analysis should be suspended or not. Their experiments show that their method reduces analysis time by 67.1% while still maintaining high-coverage of the URLs tested in their dataset.

However, contrary to the approach we propose later in this thesis, such methods do not explicitly deploy the extensive domain knowledge that has been gathered by cyber security experts over the years (e.g., in the form of TTPs from the MITRE ATT&CK framework), to create more informative features. Instead such knowledge is used to either choose the most suitable ML model for the task, or to select/transform the set of features in the most suitable way.

Despite the availability of a wide range of domain knowledge, so far not many ML-based approaches have actually exploited it, and the ones that have, did so for very specific tasks. One such task is to use domain knowledge to create better feature representations for ML models. An example of such research is the work by Bartos et al. [20], where the authors create vectorial representations of each network sample such that they are invariant to: (*i*) the scaling and shifting of the flow-level feature values and, (*ii*) the permutation of the flows in the sample. They create such a representation to design a system that is robust with respect to certain evasion techniques, such as the change in inter-arrival time between malicious flows. Each vectorial representation is then passed to a standard Support Vector Machine (SVM) [46] to distinguish between benign and malicious samples. On the other hand, research that focuses on anomaly detection, such as [53, 45, 177], computes the entropy of each captured feature and uses them to create a representation that is passed on to ML models. The intuition behind this approach is that significant traffic anomalies will cause changes in the distribution of the observable traffic behaviour (e.g., connections to IP addresses or ports), and hence will be reflected in the entropy of the features. Even if the basic intuition behind the feature representation is the same, the actual implementation varies and, further, the models used in these works are different: (*i*) Do et al. [53] use a Deep Neural Network (DNN) trained in a supervised fashion to distinguish among benign traffic, DDos attacks, PortScan attacks, P2P DoS attacks and network scan attacks, (*ii*) Cordero et al. [45] use another DNN, though trained in an unsupervised fashion, to create a normality model of network flows, which is then used to identify DDoS resource exhaustion attacks and SYN port profiling attacks, while (*iii*) Tellenbach et al. [177] first detect an anomaly and then uses a SVM to classify its type.

Other research simply uses the domain knowledge to select the best subset of features for the task at hand. Among these, one such example is the work by Bekerman et al. [24], in which the authors select 972 behavioural features across all network layers (except the physical layer) and three different protocols (DNS, HTTP, and SSL). The choice of these particular protocols was justified by the fact that they are the most commonly used application layer protocols, and subnetworks use well-configured firewalls that block connection to/from unknown protocols, alongside ports and application layer protocols that are not in the scope of the organisational policy. While Bekerman et al. explore the importance of gathering features from different networks layers and protocols, Feng et al. [61] argue for the exploitation of both static features (i.e., features that can be extracted by unpacking or disassembling the software without running it) and network level features. They focus on the detection of Android malware and argue that additionally using static features can be beneficial, since attackers nowadays use powerful code obfuscation and repackaging techniques to conceal their behaviour and evade detection.

Finally, with the advent of deep learning, many works now use the available domain knowledge to choose the most suitable topology of neural networks to apply to the task. For instance, Piskozub et al. [148] try to distinguish benign and malicious flows on a network, and, if the flow is malicious, they then try to classify the type and family of the malware. To this end, their neural model consists of: (*i*) a denoising auto-encoder, which creates a vectorial representation of the set of flows passed as input, and (*ii*) a cascade of feed-forward neural network models, each in charge of a different level of classification (i.e., the first model decides whether the flows is malicious or benign, the second one classifies the malware type, and the third one classifies the malware family).

Given the extensive literature on the topic of malware detection on networks using ML models, we give a summary of it in Table 2.1, while for a more generic overview of the methods that have been developed for malware detection and classification we refer to the surveys [8, 15, 89].

ML Method	**Work**	**Features**	**Target Class Domain**
Log. Regression	Bapat et al. [19]	Global statistic of sample (e.g., mean+std of flow duration)	Malware
AdaBoost	Khan et al. [92]	URL	Malware
Decision Tree	Wang et al. [188]	HTTP request features and TCP flow features	Malware + Family
	Bekerman [24]	Features from protocols: HTTP, SSL, DNS, TCP, IP, UDP	Malware + Family
SVM	Bartos et al. [20]	Vectorial representation of sample	Malware Sample
	Huang et al. [83]	URL	Phishing
	Tellenbach et al. [177]	Entropy of network flow features	Anomaly + Anomaly Type
Random Forest	Soska et al. [169]	Content of Web pages	Infected Websites
	Bilge et al. [26]	Flow size, time	Botnets
	Bekerman [24]	Features from protocols: HTTP, SSL, DNS, TCP, IP, UDP	Malware + Family
DNN	Piskozub et al. [148]	Network flow features	Malicious Flows + Malware Family
	Feng et al. [61]	Static features + Meta information for each packet	Malware + Category + Family
	Wang et al. [189]	URL	Malware
	Do et al. [53]	Entropy of network flow features	Benign, DDoS, PortScan, P2P DoS, Net Scan
	Cordero et al. [45]	Entropy of network flow features	Anomaly
XGBoost	Dhaliwal et al. [51]	Network-level + Host-level features	Anomaly
	Bhattacharya et al. [25]	Network-level + Host-level features	Attack Type
All of the above	Ours	Vectorial representation of sample + TTP-based features	Malware

Table 2.1: Overview of the existing state-of-the-art approaches focusing on classification of malicious traffic on networks.

2.3.4 Graph-based Approaches

A graph-based approach to malware detection and classification is another technique used in intrusion detection systems. Graphs are a useful method to represent complex data and map dependencies between distinct entities. In the context of malware analysis, they provide an effective means of representing the inherent structure, properties, or behaviour of malware samples. Graph-based approaches are also suited to the cybersecurity domain since they allow for fast retrieval of data, and their ability to attach contextual information to data allows for a more grounded inference [11]. Graph-based methods have been used widely in malware research literature to detect, classify, and mitigate malware. These include techniques used in both static and dynamic analysis systems, such as control-flow graphs [101, 27, 97, 39], system calls [4, 7, 42], and taint analysis [201].

Graph-based approaches have also been proposed for network-based malware detection and classification. For instance, Nari and Ghorbani [132] propose an automated malware classification system based on the network behaviour of malware samples. Their system represents network behaviour as high-level behavioural profiles that contain communication information such as IP addresses, ports numbers, protocols, and dependencies between network activities. These profiles are then modelled as graphs, and classified into their respective malware families based on a set of graph features.

Another technique is proposed by Milajerdi et al. [123] in their malware detection system, HOLMES. It focuses specifically on detecting Advanced Persistent Threats (APTs) in real-time by correlating tactics, techniques, and procedures (TTPs) that are used to execute various stages of an APT campaign. It also generates a high-level scenario graph (HSG) that summarises the attacker's actions in real-time, and can be used by a human analyst for a proactive response. The nodes of this HSG correspond to TTPs and the edges represent information flows between the entities involved in the TTPs. The structuring of all this information within the HSG is what allows HOLMES to detect APTs with a high degree of confidence.

Other graph-based approaches to this problem include Hercule, an automated multi-stage log-based APT attack detection system. Taking inspiration from graph analytics research in social network analysis, their system models multi-stage intrusion detection as a *community discovery problem*. They correlate entries across multiple lightweight logs to build multi-dimensional weighted graphs, and use them to detect "attack communities" embedded within them. In a similar vein, Peng et al. [144] build a malicious domain detection system using graph-based approaches. Milajerdi et al. [122] propose an APT attack detection system called ProPatrol, that uses high-level attack graphs created from kernel audit logs. They leverage the structure of malware communications to improve attack detection, identify true dependencies, and quickly pinpoint the root-cause of attacks. Liu et al. [110] propose a heterogeneous graph embedding based modularised system called log2vec that can be used for APT and insider-threat detection. It maps logs entries into a heterogeneous graph, and uses graph-embeddings to represent log entries as low-dimensional vectors. This vectorisation allows it to compare actions directly and detect anomalies. Hence, their system is not only able to identify malicious activities, but it also clusters log entries into malicious and benign groups.

More generally, network-based intrusion detection has also been modelled as a graph-theoretic problem. For instance, Najafi et al. [131] suggest modelling threat detection in Security Information and Event Management (SIEM) environments as a large-scale graph inference problem. They create a SIEM-based knowledge graph that can map the relationships between entities observed in proxy and DNS logs, and is enriched with related open source intelligence (OSINT) and cyber threat intelligence (CTI). Finally, they propose MalRank, a graph-based inference algorithm that can infer the maliciousness of a node based on its associations to other entities in the knowledge graph. They find that this system has a high detection rate that outperforms its predecessors in both efficiency and accuracy, and can detect previously unknown malicious entities. Another graph-based threat analysis technique was suggested by Shu et al. [166]—they proposed a threat intelligence graph model that could be queried by analysts for the explicit purpose of threat hunting. Other approaches include the kind proposed by Chau et al. [33] where they model malware detection as a large-scale graph mining and inference task. Nodes in this graph are a representation of users and files, while an edge indicates a particular file being present on a user's machine. The key idea behind their approach to identifying malware is locating files with low reputation. Their system computes file reputation using their Polonium algorithm, which is a fast and scalable implementation of Belief Propagation. They evaluate their method on a very large dataset containing nearly a billion nodes and roughly 37 billion edges, and find a high true positive rate (85%) alongside a low false-positive rate (1%)—thus demonstrating its feasibility as a scalable and accurate graph-based malware detection system.

2.4 Malware Classification

Malware classification is the process of identifying the group or family of malware that a particular malware sample belongs to. This includes identifying polymorphic or metamorphic versions of a given malware, and recognising new variants of malware based on existing malware. Unlike malware detection, where the structure of the malware may vary but the behaviour is usually consistent, malware behaviour in this case can be similar but not exactly the same when identifying members of a common malware family. Consequently, this is a step that takes place after malware detection and is focused solely on categorising software that is recognised as being malicious. There are a number of reasons why malware classification is useful and important in curbing the effect of malware. These include:

1. Quickly identifying the capabilities of a malware samples based on the properties of the malware in its family. When dealing with malware, being able to respond rapidly and effectively is critical to ensuring the safety of a system or network. Hence, by being able to leverage the previously acquired knowledge about a given malware family, a malware analyst can limit the effectiveness of the malware, curb the spread by blocking known propagation mechanisms and infection vectors, and deploy previously-tested countermeasures against

it. This rapid response is entirely a feature of recognising a certain malware profile and countering it based on previous knowledge, without having to analyse it first.

2. Determining new and unseen types of malware. By being able to identify those malware samples that do not belong to a known family type, analysts can focus their attention on analysing a new threat. This enables them to assess the capabilities and potential damage of the newly discovered sample, prioritise the threats posed by this malware, and consequently lead to the development of effective mitigation mechanisms for it [18]. This allows for a more effective incident response procedure, regardless of whether the threat is novel or previously-seen, thus resulting in a quicker remediation process.

3. Understanding the evolution of malware families. Grouping malware samples into families based on their specific characteristics and behaviours allows malware analysts to understand how this malware has evolved over time. This can be useful for gaining insight into the world of malware development—why is a particular malware still popular and active, what kind of infection mechanisms have been utilised by it over time, what are the kinds of vulnerabilities being exploited to launch an attack?

A number of classification techniques have been proposed to accurately and efficiently classify groups of similar malware together. In this section, we shall concentrate on classification methods for network-based malware analysis. A more comprehensive collection of malware classification techniques can be found in survey papers, such as the ones authored by Egele et al.[56], Gandotra et al. [67], or Ucci et al. [181].

One classification technique is based on the idea of observing malware's network traffic in an isolated environment to analyse potentially malicious behaviour. Rossow et al. [155] propose such a system, called Sandnet, which is capable of analysing network behaviour for long periods of time (in the order of an hour) and generating a comprehensive overview of malware network activities. This was unlike other dynamic malware analysis tools such as Anubis [22, 23] and CWSandbox [192], which provide only basic network statistics about analysed malware, and have short analysis periods (in the order of a few minutes). Their system enables them to view the kinds of network protocols being used by malware currently, and then provide an in-depth analysis for protocols that are deemed important or popular (DNS and HTTP in the case of their dataset).

Another classification technique is based on the idea of clustering malware samples based on the similarities of their network behaviour. These techniques leverage the fact that network-level activities of malware display distinct behaviours, and can uniquely provide specific insights that can be used to detect, classify, and mitigate malware [155]. Several clustering-based classification techniques [152, 132, 145, 146, 31] have been proposed that classify malware based on their network activity. One such approach was proposed by Perdisci et al. [145] which focuses specifically on HTTP-based malware. Their system clusters samples based on the structural similarity of HTTP traffic generated by malware. Their network-level analysis based on this system is able to reveal similarities between malware samples that may not be captured by current system-level behavioural clustering systems. A similar approach is adopted by Rafique and Caballero [152]—their system, FIRMA, clusters malware binaries into families and generates a set of network signatures for each family. Unlike previous methods, their system is capable of producing network signatures for malware using any network protocol. Cavallaro et al.[31] propose another clustering-based technique that can classify malware based on payload-agnostic network behaviour. Their system monitors network traffic, summarises it into a set of flows, and then clusters them hierarchically on the basis of a given set of network features and periodic behaviour exhibited by malware. Their approach has several advantages: it can handle encrypted traffic since it does not inspect traffic content, it does not require observing noisy behaviour, and it has a low false-positive rate. These properties make their network behaviour detection models ideally suited for deployment in a real-world settings.

Another popular technique that is used for classification on the basis of network-level behaviour is using n-gram analysis. This is a widely-used method in host-based static and dynamic analysis systems, and several techniques [193, 127, 120, 10] have been proposed that leverage n-grams at the network level. Wressnegger et al. [193] analyse n-gram models used in intrusion detection systems. Through their analysis, they develop criteria to help ascertain whether some network data is better suited for either classification or anomaly-detection using n-gram models. Mohaisen et al. [127] propose Chatter, a malware classification system that leverages the order in which high-level system events occur. It captures and analyses these order-based features of network communications using n-grams. Their experiments show an 80% accuracy when dealing with a dataset containing 3 malware families, which increases to 95% when combined with a baseline classifier of non-ordered features. Mekky et al. [120] propose a supervised machine learning based system that can classify malware by observing the mixed network traffic of an infected host. They use n-gram frequencies as features in their classifier in order to preserve the order of network events. This system differs from Chatter because it uses coarse-grained network events instead of the fine-grained events utilised by Chatter. Secondly, it separates network traffic into malware and background traffic using Independent Component Analysis (ICA) [44], whereas Chatter assumes that only clean malware traffic is being used. These techniques improve the performance of this system's classifier to roughly 95%, a significant improvement over existing systems. AlAhmadi and Martinovic [10] proposed MalClassifier, a network traffic based malware classification system that uses n-gram analysis for network flow sequence mining. It uses n-grams to sort network flows into groups of n consecutive flows (or n-flows), which are then used as features to train a supervised model that can classify unseen n-flows into a given malware family. Their system improves on previous research by using non-payload features (and hence being resilient to encrypted traffic), adapting to malware behaviour changes, analysing malware without the need of a sandbox, and being protocol independent. This system design enables MalClassifier to achieve an F1-score of 96% for family classification in their dataset.

Every accomplishment starts with the decision to try.

— John F. Kennedy

3 Extensible and Explainable Malware Detection using TTPs

Contents

3.1 Introduction 42
3.2 System design 45
3.2.1 Data Ingestion and Graph Construction 46
3.2.2 TTP Detection 53
3.2.3 TTP-based Classification System 55
3.3 Evaluation 58
3.3.1 TTP Extraction 59
3.3.2 Classification performance 61
3.4 Discussion 67
3.5 Related Work 70
3.6 Limitations 71
3.7 Conclusions 71

3.1 Introduction

As we have seen in the previous chapter, the capabilities and sophistication of malware authors has increased greatly in recent decades and novel attacks are continuously being developed. According to the 2021 Data Breach Investigation Report by Verizon, nearly 20% of data breaches are a result of system intrusions, of which over 80% are due to malware [21]. This, in conjunction with the increasing number of networked devices, has made it commensurately harder to defend both networks and devices from malicious attacks. To keep up with these rapidly evolving threats, more and more sophisticated malware detection systems are being designed, many of which are based on machine learning (see, *e.g.* [105, 69, 164]). Though such systems attain impressive results, they are often conceived having effectiveness as the main, if not only, requirement. As a result, the effectiveness of such systems, especially when based on deep learning models, often comes with (*i*) poor extensibility, being monolithic systems they are very difficult to adapt and/or extend to other settings, and (*ii*) poor explainability, since it is not possible for humans to understand the reasons behind the model's predictions, and also given their poor interpretability, this due to their inner complexity which makes it impossible to link the result of detection with its root cause. This is unfortunate: extensibility, explainability, and interpretability are important properties of any system in general, and in particular in the field of cyber security. Indeed, malware detection systems need to be continuously updated in order to deal with novel attacks, which, when detected, often need to be further analysed and inspected by domain experts (see also [16]). Despite this, systems for malware detection are usually not designed to be extensible, and the vast majority of them are neither explainable nor interpretable.

In this chapter we show how it is possible to design an extensible and explainable yet effective network-based malware detection system. This system, called RADAR, (*i*) maintains an internal graph-based representation of the network in order to effectively reason about the network traffic generated by a given malicious/benign sample (hereby referred to as just *sample*), (*ii*) exploits TTPs from the industry-standard MITRE ATT&CK framework which describes post-compromise adversarial behaviour, to unequivocally identify and classify the potential malicious network flows exhibited by each sample, (*iii*) utilises a dedicated decision tree for each TTP to label each network flow as potentially malicious, and, (*iv*) combines the results obtained for the different TTPs in order to finally label each sample as either benign or malicious. As a result, it is relatively easy to extend the system with new components for detecting novel malicious behaviour using an updated set of TTPs, and/or modify existing components in order to broaden their capabilities using a combination of existing TTPs. Further, thanks to the joint use of TTPs and decision trees, RADAR is both an explainable and interpretable system, being able to output in human-understandable terms: (*i*) which high-level TTPs (and not just the low-level features) are being used in the detected malicious sample, and (*ii*) why a given sample has been labelled as malicious. Additionally, the internal graph-based representation of the network, together with the exploitation of TTPs, provide the means to give valuable aid to system analysts in their evaluation of the reported malicious activity.

We evaluate RADAR on a very large dataset comprising of 2,286,907 malicious and benign samples, representing a total of 84,792,452 network flows. The experimental analysis confirms that the proposed methodology can be effectively exploited: RADAR's ability to detect malware is comparable to other state-of-the-art systems' capabilities, as reported in the literature (see, *e.g.*, [112]). For instance, RADAR's performance as judged by an AUC score of 0.87 is similar to what has been reported in the very recent paper by Dyrmishi et al. [54]. To the best of our knowledge, RADAR is the first TTP-based system for malware detection that uses machine learning while also being extensible and explainable.

The rest of the chapter is structured as follows. Section 3.2 provides a detailed description of the RADAR design, while also presenting some of the technological hypotheses and choices that guided its implementation. RADAR is then evaluated in Section 3.3 according to different metrics and in various settings. Section 3.4 contains a discussion about the results of the development and testing of RADAR. Section 3.5 compares our system to relevant related work. We end the chapter with a section on the conclusions from designing and evaluating such a system.

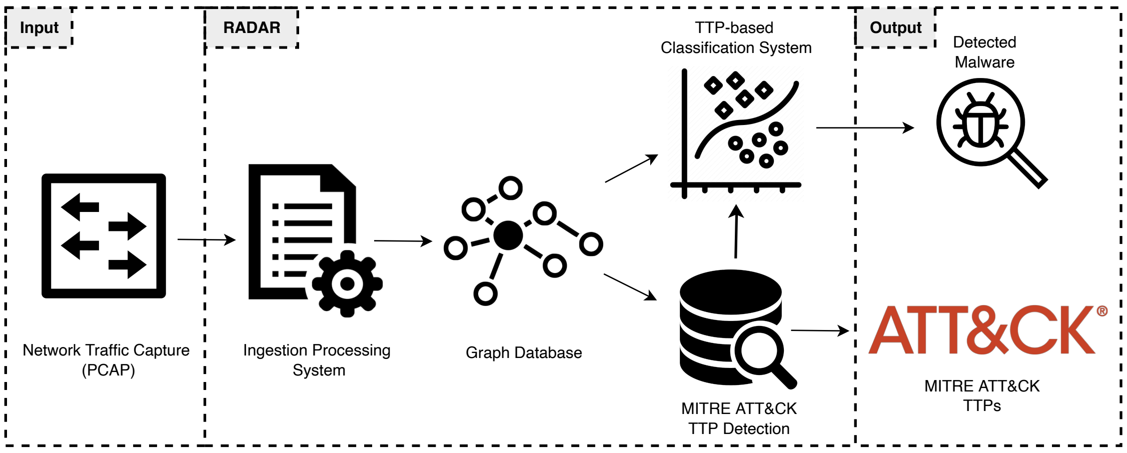

Input
RADAR
Output
Network Traffic Capture (PCAP)
Ingestion Processing System
Graph Database
TTP-based Classification System
MITRE ATT&CK TTP Detection
Detected Malware
ATT&CK®
MITRE ATT&CK TTPs

Figure 3.1: RADAR system overview.

3.2 System design

RADAR is a system for network traffic analysis and malware detection, designed to be both extensible and explainable. Extensibility is a well known software engineering and systems design principle that enables future growth and adaptation. In the field of cyber security, and malware analysis in particular, extensibility is a necessary criterion to design systems that can adapt in the face of novel and constantly-evolving attacks. Explainability has recently emerged as an important requirement for ML and AI systems. As stated in [78], explainability is essential for users to effectively understand, trust, and manage powerful artificial intelligence applications in every application domain, including cyber security (see, e.g., [16, 30]). Explainability calls for producing high-level outputs which can be rooted back to the inputs, i.e., for exploiting ML interpretable models for the classification of malicious activities. With such goals, RADAR has been designed to

1. be extensible, given its modular software architecture in which different TTPs are detected and classified by different ML engines, each of which can be easily extended and/or modified independently from the others, and

2. provide high-level explanations, given its ability to classify malicious activity using TTPs from the industry-standard MITRE ATT&CK framework, and its use of interpretable decision trees for this classification.

RADAR's data pipeline is designed to be able to take arbitrary network packet captures corresponding to a sample as input, and output:

1. the relevant matching TTPs for that sample, and
2. a prediction as to whether the sample is malicious.

This enables RADAR to be employed as a 'plug-and-play' technology wherein a packet capture can be sliced from a larger flow of network traffic and analysed. To further facilitate integrating RADAR into existing security infrastructures, it has been designed and implemented to also satisfy the following requirements:

1. Be lightweight and as efficient as possible to enable the storage and possible retention of large amounts of data.
2. Be content agnostic, *i.e.*, be based only on the analysis of the metadata, thus also working on encrypted data payloads and not violating possible privacy regulations that would otherwise limit the applicability of the system.
3. Have the ability to analyse arbitrary network captures, *e.g.*, independent of traffic type, protocol, or network topology.

The components of RADAR include an Ingestion Processing System that constructs a graph database of malware samples, a TTP Detection Engine based on the MITRE ATT&CK framework, and a TTP-based Classification System. We show an overview of our system architecture in Figure 3.1.

In the following subsections we shall cover each of the system components.

3.2.1 Data Ingestion and Graph Construction

The first stage in our data pipeline is the Ingestion Processing System that takes as input network traffic captures (PCAPs), and processes them into a format that is suitable for graph construction. The possible data formats to implement the processing of packet captures broadly fall in one of the following three categories:

1. PCAPs. PCAPs contain the most granular level of detail with regards to extracting meaningful information. Given that our system input format is already PCAP, this would reduce one layer of processing and increase system efficiency. However, since these are raw network captures, these are large files that contain a lot of information, not all of which is required by our system. As a result, storage costs are likely to be higher and processing times per file would increase. In addition, PCAPs contain complete packet dumps, including data payloads and all other metadata, which could potentially cause ethical, legal, and privacy issues.

2. NetFlows. NetFlows have also been used in a number of different existing solutions for network traffic analysis, as well as academic papers [26, 74]. Many Security Operations Centres (SOCs) also use NetFlows for the purpose of network traffic analysis. Their key limitation is the lack of detail about the underlying data. NetFlows are only able to store basic information (Egress and Ingress port, protocol, and interface) about the network traffic. Due to the lack of information available from these flows, the analysis that can be performed on them is more shallow. This lack of data means that the storage of NetFlows is compact—it also means that the processing is faster than on comparable storage formats. Additionally, they are privacy-preserving since they do not contain any data payloads and rely mostly on metadata.

3. Network logs. Network logs, like those generated by Zeek, are somewhere between PCAPs and NetFlows in terms of the granularity of data they contain. Zeek in particular has been used in academic research and existing solutions to detect malware, including the extraction of MITRE ATT&CK techniques. These logs are not comparable to PCAPs in terms of precision, but contain more information than a typical NetFlow while maintaining some of the storage and performance benefits. They are also content agnostic and thus preserve the privacy of the underlying network communications. However, these logs are typically application-specific and hence have no standard format.

Field	**Property**
Flow Hash	The hash associated with each unique flow.
Sample Hash	The hash of the sample from which the flow is derived.
Unique ID	A unique ID associated with each flow.
Dataset	The dataset to which a flow belongs.
Source Port	The port from which the flow originates.
Destination Port	The port on which the flow connects to.
Start time, end time	Flow start or end time.
Duration	Flow duration in seconds.
Protocol	IP protocol identifier in decimal format.
Entropy	The Shannon Entropy for the flow payload.
Applabel	The application label, as identified by YAF.
Source IP, Dest. IP	Source and destination IPv4/IPv6 address.
Type, Code	ICMP type or code in decimal format.
Isn, Risn	Forward or reverse TCP sequence number.
Flags	4 properties representing various TCP flags.
Tag, Rtag	802.1q VLAN tag in forward/reverse direction of flow.
Pkt, Rpkt	No. of packets in forward/reverse direction of flow.
Oct, Roct	No. of bytes in forward/reverse direction of flow.
RTT	Round-trip time estimate in milliseconds.
End-reason	Reason for termination of flow.

Table 3.1: Flow properties.

Additionally, they are wholly dependent on the application that generates them (e.g. Zeek), which forces the introduction of the corresponding software that creates these logs into the overall system pipeline, which may not be desirable from the perspective of security or flexibility.

Among the above network data formats, we selected NetFlows which rely solely on metadata, thus meeting the requirements of being content-agnostic and lightweight. In particular, we opted for a specific type of NetFlow, Yet Another Flowmeter (YAF) [86]. YAF generates bidirectional flows in IPFIX standard [43], which is based on the Cisco NetFlow V9 export protocol. Our primary motivation for selecting YAF is that it is designed for high performance and scalability across large networks. By design, it also takes into consideration the balance between capturing information and maintaining privacy on the network, and takes a hybrid approach between typical NetFlows and PCAPs. It is also more feature-rich than the typical NetFlow or network log, since it contains support for functionality such as application labelling (*i.e.*, port independent protocol checking) and entropy analysis (to determine if traffic is encrypted or compressed). The complete list of features, including those extracted using YAF, are presented in Table 3.1.

As can be seen from the table, these features are a combination of properties that we can extract from YAF (such as IPs, ports, and protocol) and custom features (such as flow hash, sample hash, and UID) that enable us to analyse a sample at different levels of granularity. For instance, a flow hash is computed on the basis of certain key values of a flow [`flow_hash: hash(src_ip, dest_ip, src_port, dest_port, protocol)`], and is not necessarily unique in the larger set of flows, whereas UID is unique per flow by design. Combined with the sample hash, this allows us to track similar flows and unique flows of interest, both within a given sample and in completely separate malware samples. This variation in granularity and aggregation helps make our TTP detection engine more robust in identifying sequences of flows that represent malicious behaviour.

Having selected and created the relevant features for our network traffic, we then map them into a graph, which represents the network traffic we want to analyse. Formally, we create an annotated graph $G = (V, A, E)$, where:

- V is the set of vertices. Each vertex represents a host and is uniquely identified by its IP address in the network traffic.
- A is the set of edge annotations. Each annotation corresponds to an instantiation of the set of properties listed in Table 3.1.
- $E \subseteq V \times V \rightarrow A$ is the set of directed edges. Each edge corresponds to a flow between the two hosts.

Figure 3.2 highlights this graph schema with the help of an example set of flows from a given sample. We choose this generalised form of graph entities and relationships precisely because it is applicable to all network traffic, and makes

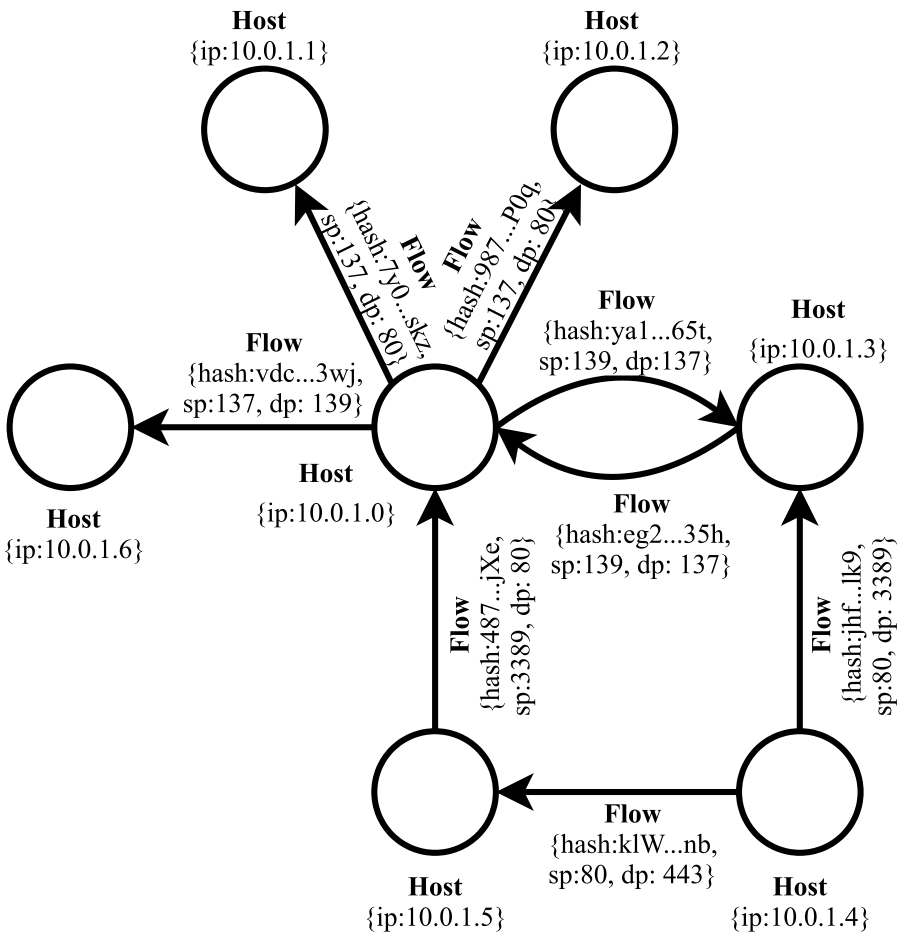

Host {ip:10.0.1.1}
Host {ip:10.0.1.2}
Flow {hash:7y0...skz, sp:137, dp: 80}
Flow {hash:987...P0q, sp:137, dp: 80}
Flow {hash:ya1...65t, sp:139, dp:137}
Host {ip:10.0.1.3}
Flow {hash:vdc...3wj, sp:137, dp: 139}
Host {ip:10.0.1.0}
Host {ip:10.0.1.6}
Flow {hash:eg2...35h, sp:139, dp: 137}
Flow {hash:487...jXe, sp:3389, dp: 80}
Flow {hash:jhf...lk9, sp:80, dp: 3389}
Flow {hash:klW...nb, sp:80, dp: 443}
Host {ip:10.0.1.5}
Host {ip:10.0.1.4}

Figure 3.2: Graph schema for network traffic.

writing TTP detection logic independent of network protocol and packet formats. Having a universal form of expressing network data is also much more desirable from the perspective of easily extending and integrating the system.

We express the graph schema explained above in our system using the Neo4j Graph Database [133], which we can query using the Cypher query language. Features from the network captures are collated and processed together in a format that conforms with our schema. Subsequently, all flows associated with a given malware sample are inserted in this form into our graph database, at which point they can be analysed. The analysis of the data in the database takes advantage of its graph structure when querying the underlying network traffic to detect various host and flow configurations. We demonstrate this point by means of a simple example presented below.

Example 3.2.1 Let M be the number of flows in our dataset and d be the maximum out-degree among all the nodes. Suppose that we want to determine for a given sample, the total number of hosts a that when communicating with another host b, also result in b establishing a connection with another host c.

This network traffic is represented as a graph in RADAR, and using a Cypher-based notation, we write a query for this graph as follows:

```
MATCH (a:Host)-[f1:Flow]->(b:Host)-[f2:Flow]->(c:Host)
WHERE f1.sample_hash = f2.sample_hash
RETURN COUNT(a, b, c);
```

The first matching operation for $a \rightarrow b$ requires traversing all possible flows, and has time complexity $O(M)$. The second match operation, $b \rightarrow c$, on the other hand is a constant time operation since only the existence of an outgoing edge from b needs to be established. This is repeated at most d times, for each of the outgoing flows from b. The remaining filtering operations in the `WHERE` clause operate on a subset of the remaining flows, and are thus guaranteed to have an upper bound of $O(M)$. Hence, owing to the graph representation, the overall time complexity of this query is $O(Md)$.

sample_hash	sip	dip	sp	dp	flow_hash	uid	...
xyz	10.0.1.10	10.0.1.1	88	80	eg123ya	47def75	...
abc	10.0.1.1	10.0.1.2	88	80	fm849ag	09pqr23	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Table 3.2: Tabular representation of network traffic data.

Now let us consider the case where the same network traffic data is stored in a tabular format, such as in a standard relational database (RDB), as shown in Table 3.2. Each row in the table represents a flow, and each column denotes the flow properties as described in Table 3.1. Then, using a SQL-based notation, we write this query as follows:

```
SELECT COUNT(DISTINCT ft1.sip)
FROM flow_table AS ft1, flow_table AS ft2
WHERE ft1.sample_hash = ft2.sample_hash AND ft1.dip = ft2.sip;
```

Here, the Cartesian Product of the table with itself in the `FROM` clause, results in a time complexity of $O(M^2)$. The overall complexity of the query remains $O(M^2)$ since the subsequent `WHERE` and `SELECT` clauses operate on a subset of the result of the Cartesian Product. □

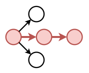
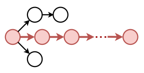
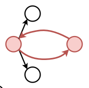
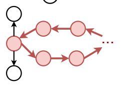
Operation	**Visualisation**	**Complexity (Graph)**	**Complexity (RDB)**
Detection of all traffic traversing at least 3 hosts		$O(Md)$	$O(M^2)$
Detection of all traffic traversing at least $k > 3$ hosts		$O(Md^{k-2})$	$O(M^{k-1})$
Detection of all bidirectional flows between 2 hosts		$O(Md)$	$O(M^2)$
Detection of all bidirectional flows traversing $k > 2$ hosts		$O(Md^{k-1})$	$O(M^k)$

Table 3.3: From left to right: examples of possible operations on network traffic data, their graph visualisations, time complexity of their associated query if data is represented as a graph, and time complexity of their associated query if data is represented using a relational database.

The benefits in query performance in terms of time complexity given by the graph representation is not limited to the query in Example 3.2.1. This and other similar operations useful in analysing network traffic data can be generalised further and the overall benefit of the graph-based modelling can be realised much more concretely. We present some of these operations, their graph visualisation, and their graph-based and RDB-based time complexities in Table 3.3. As can be seen in the table, the graph representation used by RADAR is more efficient than standard RDB-based modelling.

Tactics	Techniques	Malicious	Benign	Class.
Recon.	T1590 - Gather Victim Netw. Inf.	4	4	✗
Cred.Access	T1557.001 - Man-in-the-Middle	7	0	✗
Discovery	T1124 - System Time Discovery	1,645,252	103,461	✓
	T1135 - Network Share Discovery	8224	41	✓
Lateral Movement	T1021.001/4 - Remote Services(RDP/SSH)	666	20	✗
	T1550.003 - Use Alt. Auth. Material	241	1	✗
	T1563.001/2 - Rem.Serv.Sess.H.(RDP/SSH)	3	0	✗
	T1570 - Lateral Tool Transfer	1	0	✗
Command and Control	T1071 - Application Layer Protocol	1997	15	✓
	T1090 - Proxy	965	2	✗
	T1105 - Ingress Tool Transfer	938	163	✓
	T1571 - Non-Standard Port	1,970,237	119,501	✓
Execution	T1053 - Scheduled Task/Job	10	5	✗

Table 3.4: Supported MITRE ATT&CK TTPs. The first column shows the tactical goals of the adversary. The second column shows the (sub-)techniques used to achieve them. The third and fourth columns indicate the absolute number of malicious and benign samples per TTP in our dataset. The last column indicates whether it is feasible to train a classifier for the given TTP.

3.2.2 TTP Detection

The TTP Detection Engine analyses all flows in the graph database and executes detection rules on the graph entities and relationships to identify any matching MITRE ATT&CK technique. The first two columns of Table 3.4 show the complete list of TTPs that our system currently supports (notice that T1021 and T1563 have two sub-techniques each).

The set of rules utilised by our TTP Detection Engine has been selected being a good representative of the standard malicious activity carried out on networks, diverse enough in order to stress the ability of the system to detect different kinds of attacks. The TTPs we consider broadly falls into two categories:

1. **Feature-based detection rules.** This type of detection relies on specific features of the flows described in Table 3.1.

2. **Heuristic-based detection rules.** This type of detection relies on the analysis of multiple flows and uses heuristics based on their properties.

We demonstrate the utility of both types of rules and the specific logic required with the help of two examples.

Example 3.2.2 For the feature-based rules, we consider the MITRE ATT&CK technique T1571 (Non-Standard Port). This technique is part of the "Command and Control" (C&C) tactic and is applicable to the case when a host attempts to communicate with another host using standard network protocols but on a non-default port. Malware can attempt such a connection for several reasons—to communicate using a random port, evade firewall rules blocking certain default ports, or to find an open port for the purpose of establishing a C&C channel, among others. In order to detect this technique we rely on the features of the flow the two hosts use to communicate with each other. Specifically, we compare the network protocol in the flow with the port(s) used during communication and determine if these correspond to their defaults. In order to do this, we utilise YAF during the Ingestion Processing phase to determine the application layer protocol of every flow independent of the port used. This feature is extracted and stored in our graph as the flow property `applabel`. Our detection logic maintains a list of the 50 most popular application layer protocols and their corresponding default ports, and compares these to the features of `applabel` and `port` present in every flow. A mismatch between these features is a positive match for T1571.

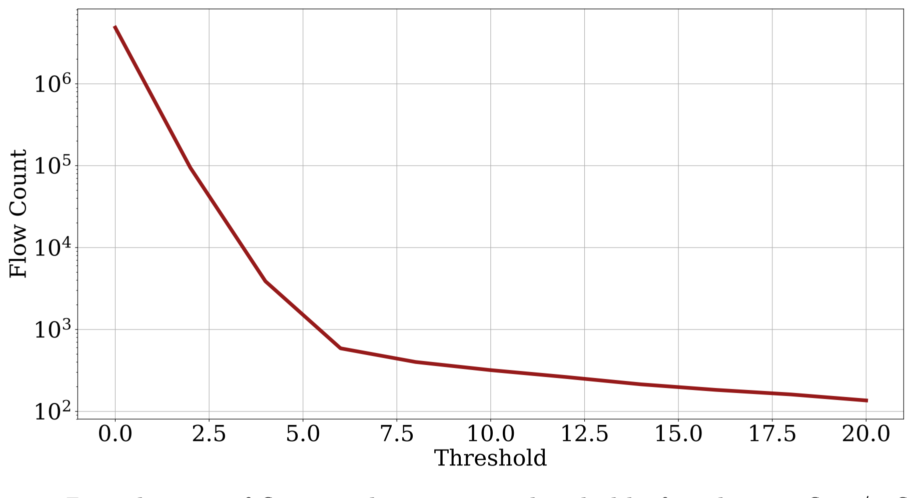


Figure 3.3: Distribution of flows with a varying threshold of packet outflow/inflow ratio, with a logarithmic Y axis.

Example 3.2.3 For the heuristic-based rules, we consider the MITRE ATT&CK technique T1071 (Application Layer Protocol). This technique is also part of the C&C tactic, and is applicable when an application layer protocol is used to hide malicious traffic. Specifically, we attempt to identify this technique by classifying flows as anomalous on the basis of their incoming and outgoing traffic relative to other flows. For each malware sample, we isolate its flows and establish a mean packet outflow to packet inflow ratio through their directed edges, and determine a unique baseline for each sample. Then by defining a threshold over which there is far more data egress than ingress as compared to the baseline, we identify flows and hosts that are potentially communicating via a C&C channel or exfiltrating data from the network. Figure 3.3 shows the distribution of this ratio against a variable threshold over a subset of our data. This threshold is designed to be dynamically adjusted based on the distribution of the network traffic and the characteristics of what is considered "normal" in a given network or environment.

At the end of this process, the system outputs a list of flows and their corresponding TTPs. This information is further aggregated by combining all the flows belonging to the same sample and then passed on to the TTP-based Classification system.

3.2.3 TTP-based Classification System

The TTP-based Classification System checks whether the flows in each sample exhibit malicious behaviour when compared to other flows matching the same TTP, and then flags a sample as malicious if "enough" flows exhibit malicious behaviour. The TTP-based Classification System works at deployment time following the steps in Figure 3.4.

As shown in the figure, the TTP-based Classification System takes as input: (*i*) the flows' properties (as listed in Table 3.1), and (*ii*) the TTPs detected in each flow, as returned by the TTP Detection Engine. Following a "divide-and-conquer" approach, it consists of one base classifier for each analysed TTP. This choice allows us to compare the behaviour of each flow with only those flows that match the same TTPs.

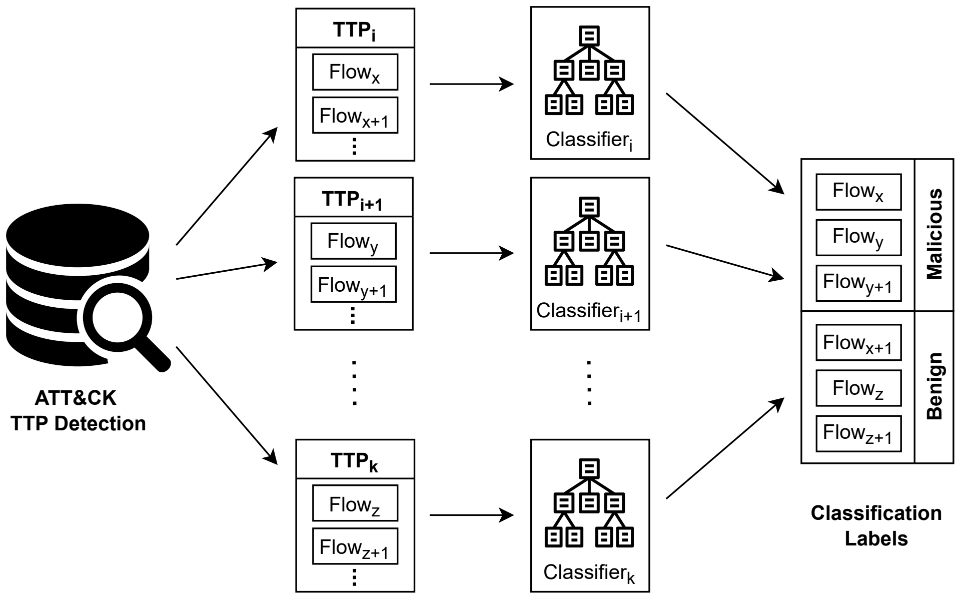


Figure 3.4: TTP-based Classification System Pipeline

For each flow f, we create a vector x containing all the features listed in Table 3.1. Then, x is passed to all the base classifiers that have been trained on the flows that matched the same TTPs as f. Thus if, for example, f matched TTP$_i$, TTP$_{i+1}$ and TTP$_k$ (as in the Figure) then x will be passed just to the ith, $(i+1)$th and kth classifiers. Each of the selected classifiers then independently decides whether f presents malicious behaviour when compared to the other flows flagged with the same TTP. Notice that in general any classifier can be used for this task. However, in order to preserve the interpretability and explainability of the system, we opt to use decision trees [196]. Indeed, a decision tree is a transparent model whose predictions can be easily explained via logic rules. This is not true for most classifiers (*e.g.*, based on artificial neural networks) which are often completely opaque models whose predictions are difficult to interpret. As a result of this choice, it is possible to see exactly which network properties are used to make a classification decision, and this information can be quite useful to SOC analysts by helping them triage alerts in a more informed manner.

Once we have the prediction for each flow in the sample, we have to find a way to aggregate them and decide whether the sample is malicious or not. To this end, we devise three policies:

1. P1: the first policy relies on n_t, the number of unique TTPs matched by the malicious flows in the sample. If $n_t \geq \theta_t$ then the sample is considered malicious. θ_t is a parameter that the user can tune.

2. P2: the second policy relies on n_f, the number of malicious flows in the sample. If $n_f \geq \theta_f$ then the sample is considered malicious. θ_f is a parameter that the user can tune.

3. P3: the third policy relies on p, the percentage of malicious flows in the sample. If $p \geq \theta_p$ then the sample is considered malicious. θ_p is a parameter that the user can tune.

The first policy, P1, is devised on the ground of the hypothesis that the more unique TTPs a sample maliciously uses, the more likely it is that the sample is engaged in malicious activity. However, this policy can become quite strict very quickly, as it is unlikely that the maliciousness of a sample will increase linearly with the number of TTPs after a certain point. We thus devise an alternate policy, P2, which is based on the number of malicious flows in a sample. The benefit of such a policy is that it does not rely solely on the presence of unique TTPs, and instead takes into consideration the fact that the overall repeated malicious usage of TTPs might be a good indicator of the sample's maliciousness. However, this policy's reliance on absolute values makes it susceptible to missing samples with lower numbers of malicious flows. Hence we devise our third policy, P3, which takes into account the percentage of malicious flows in the sample. It has similar benefits as P2 by virtue of utilising the repeated usage of TTPs and not relying solely on the presence of unique TTPs, however it uses the relative proportion of malicious flows in the overall sample to determine maliciousness. Notice that integrating other custom policies is quite straightforward, *e.g.*, assigning different weights to the TTPs and/or to the multiple uses of the same TTP. Thus, by designing and tuning policies for specific contexts, further performance gains can be obtained.

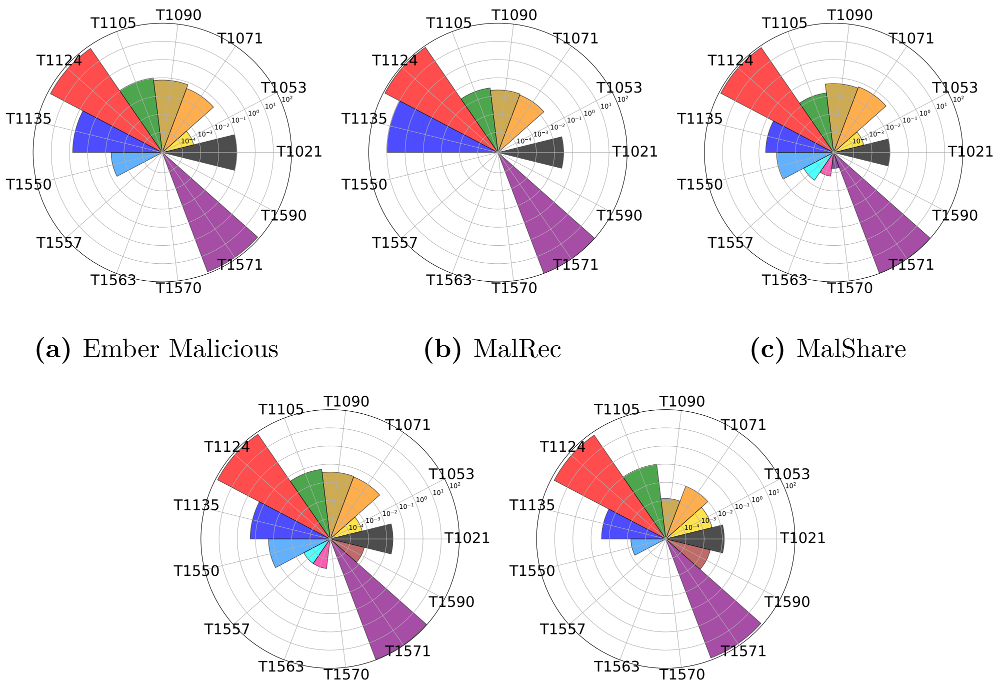
Dataset	**Samples**	**Flows**
Ember Malicious [13]	435,741	26,567,527
MalRec [160]	37,763	12,273,502
MalShare [150]	1,268,923	32,310,907
VirusShare [183]	595,098	12,217,493
Ember Benign [13]	155,432	1,423,023
Total (unique)	2,286,907	84,792,452

Table 3.5: Overview of all the datasets. The first four datasets contain only malicious samples, while the last one contains only benign samples.

3.3 Evaluation

We take a similar approach to many well-known systems in the literature when evaluating RADAR, wherein we test our detection capabilities against real-world malware datasets [142]. In doing so, we use an especially large dataset of network flows collected from malware samples for testing. We execute our samples within a Cuckoo sandbox on VirusTotal. All samples are allowed to freely create any network connections they wish, and the network traces of their actions are collected. Each sample is executed for two minutes within the Cuckoo sandbox, as research has shown this is sufficient for observing the majority of a sample's behaviour [102]. The resultant PCAPs corresponding to each sample are then given as input to RADAR in order to commence analysis.

Table 3.5 shows the complete list of datasets, along with their properties, that we use to test our system. Ember is an open dataset of hashes of malicious Windows executables created as a benchmark for machine learning models. It contains 3 types of hashes: malicious, benign and unlabelled. For our purposes, we utilise both the malicious hashes (as Ember Malicious) and the benign hashes (as Ember Benign) in our dataset. MalRec is a small dataset created as a result of a running malware on a dynamic analysis platform, which was collected over a two-year period. MalShare is a community-driven open repository of malware samples, which features over a million hashes, from which the corresponding dataset is derived. Similarly, VirusShare is a repository that aims to help security researchers analyse selected



(a) Ember Malicious **(b)** MalRec **(c)** MalShare

(d) VirusShare **(e)** Ember Benign

Figure 3.5: Percentage of samples matching various MITRE ATT&CK TTPs across each dataset, with a logarithmic Y-axis.

strains of malware, and the source of our corresponding dataset. We select these datasets as they are representative of modern commodity malware and allow for the reproducibility of results. It is important to note that we do not utilise the entirety of each dataset in our experiments, since we filter out samples that do not exhibit any network activity. Overall, we analyse a total of 84,792,452 flows from 2,286,907 malicious and benign samples.

3.3.1 TTP Extraction

3.3.1.1 Detected TTPs and Significance

We deploy RADAR against the datasets shown in Table 3.5 to detect the TTPs our system supports. Table 3.4 shows the instances of the various MITRE ATT&CK TTPs that RADAR is able to detect in both our malicious and benign datasets and the total number of samples for which the corresponding TTP is detected in the datasets. As seen in Table 3.4, we are able to detect nearly every type of

MITRE ATT&CK TTP that RADAR supports in our dataset, though there is large variance in their frequency. We also analyse the frequency of distinct TTPs per malware sample in our datasets. We observe that only 8.62% of samples are using no detectable techniques while for almost all the other samples RADAR is able to detect 3 or more techniques, indicating that our broad selection of TTPs is significant being able to identify malicious activity within the vast majority of malware samples within our datasets.

3.3.1.2 Variance of TTPs Across Datasets

We further explore the distribution of different techniques across all our datasets. Figure 3.5 shows the percentage of samples matching each MITRE ATT&CK TTP in every dataset. We find that certain TTPs have a high prevalence in all our datasets (*e.g.*, T1124 and T1571), while others occur with varying frequency depending on the specific dataset. These heterogeneous distributions are to be expected since each of these datasets is from different sources, and indicate that for the best performance, the system has to be tuned depending on the specific operational context. While the individual prevalence of certain techniques in our malware datasets shows the unique composition of the types of malware in each dataset, the frequency of occurrence of certain techniques in our benign dataset showcases some of the differences between the TTPs utilised by malicious and benign samples. For instance, T1550 (Use Alternate Authentication Material), T1557 (Man-in-the-Middle), and T1563 (Remote Service Session Hijacking) can be found in the MalShare and VirusShare datasets, while of these only T1550 is detected in Ember Malicious, while MalRec contains none of these TTPs. On the other hand for instance, the occurrence of T1090 (Proxy) is relatively infrequent in the benign dataset when compared to the malicious datasets. This would logically follow since benign samples are unlikely to utilise proxy services like Tor as frequently as malware samples.

3.3.1.3 Malicious Usage of TTPs

Our results highlight that techniques such as T1571 (Non-Standard port), T1124 (System Time Discovery) and T1135 (Network Share Discovery) are the most frequently occurring across all our malicious datasets. Other techniques used by malware in significant numbers include T1071 (Application Layer Protocol), T1021 (Remote Services), T1105 (Ingress Tool Transfer), and T1090 (Proxy). Whilst we discover evidence of other TTPs being used by malware, they are far less prevalent in their usage, but still indicate usage by a non-negligible amount of samples. The most commonly occurring TTP in our dataset is T1571 (Non-Standard Port). This shows a high prevalence of malware making use of ports that are not the default original source or destination port for a given application protocol. Whilst this technique has been written about in various papers and threat intelligence reports by antivirus firms (see, *e.g.*, [176, 152]), our pervasive detection of T1571 in all the different datasets affirms that this is a technique used by a large variety of different malware samples 'in-the-wild'.

3.3.1.4 Benign Usage of TTPs

We also observe a high number of TTPs being detected in our benign dataset. Specifically, T1571 (Non-Standard Port) and T1124 (System Time Discovery) are by far the most frequently occurring TTPs in our benign dataset. This is not surprising considering that benign software can be easily responsible for both these behaviours: (*i*) many applications can use non-standard ports to establish connections via commonly-used protocols (such as connecting via HTTPS on a port other than 443) and, (*ii*) several applications rely on having an accurate measure of time (say by contacting an Network Time Protocol server) to function correctly. This highlights the challenge of using solely TTPs to determine the maliciousness of a sample.

3.3.2 Classification performance

In this subsection, we evaluate how well RADAR can distinguish between malicious and benign use of TTPs on the dataset obtained by merging the datasets in Table 3.5.

3.3.2.1 Experimental Setup

Dataset Creation per TTP. Given our a dataset, we first need to assess whether enough flows/samples are matching a TTP to train the corresponding classifier. The results of this analysis are reported in the last column of Table 3.4. If we do not have enough flows/samples matching a TTP to train the corresponding classifier, we just classify each flow matching the TTP as malicious, and then we use the desired policy to conclude whether the sample is malicious or not.

For those TTPs for which training a classifier is feasible (*i.e.*, TTPs: T1071, T1105, T1124, T1135 and T1571) we then need: (*i*) a "benign"/"malicious" label for each flow, and (*ii*) a training, validation and test set for each classifier. Regarding the labels, we simply assign a flow the label "malicious" (resp. "benign") if the sample to which it belongs is malicious (resp. benign). Regarding the creation of the datasets splits, we need to make sure that: (*i*) flows from the same sample do not belong to different splits, (*ii*) flows matching multiple TTPs belong to the same split for the different classifiers. To achieve the above we perform two steps. Firstly, we split the dataset obtained as the union of the datasets in Table 3.5 in train, validation and test sets, such that: the train set contains 60% of the samples, the validation set 10% of the samples, and the test set 30% of the samples. Secondly, for each flow f in the training set and for each TTP T, we check whether f matched T, and if that is the case, then f is added to the training set of the classifier for the TTP T. The same procedure is then repeated for all the flows in the validation and test set. This ensures that the splits for each classifier retain the desired properties.

Pre-processing. One of the advantages of decision trees is that they do not require much data pre-processing. We thus need to perform only two steps: (*i*) discard those features that we know *a priori* will not help the classification, and (*ii*) transform all the non-numerical features to numerical ones.

Regarding the first step, we drop the features "Flow Hash", "Sample Hash" and "Unique ID" as they are simply identifiers of the flows and/or samples. We also drop the feature "Dataset" in order to avoid the decision trees to incorrectly learn to classify flows on the ground of which dataset they are drawn from. Then, we drop the features “Source IP”, “Start Time”, “End Time” because the samples are executed in a sandbox, and thus, while they might be useful in practice, in our dataset they carry no useful information. Finally, we also drop the features “Isn” and “Risn” as they are categorical features with too many unique values (63,796 and 8,229,092, respectively).

Regarding the second step, the features “End-reason”, “Protocol”, and the four features representing the TCP flags are made of categorical values, and thus we transform them in one-hot encoded vectors. Different considerations are taken for the feature “Destination IP”, for which we want to retain the ordinal information inherent of IP addresses. Thus, each IP address is mapped to a 32-bit integer in the following way: given an IP address $w.x.y.z$, the final *dip* feature is given by: $(w \times 256^3) + (x \times 256^2) + (y \times 256^1) + (z \times 256^0)$.

Decision Tree Training. Each decision tree is then trained on the training set of the relevant TTP by maximising the Gini gain at each splitting step. In order to alleviate the imbalance problem across our malicious and benign datasets, we associate weights to each of the two classes that are inversely proportional to the class frequency in the training set. Further, for each decision tree associated to a TTP, we optimise two hyper-parameters: (*i*) the maximum depth of the decision tree, and (*ii*) the minimum number of samples in each leaf. For the maximum depth we try all the possible depths between 1 and 50, together with the option of not setting a maximum depth at all, while for the minimum number of samples we test values 1, 100 and 1,000. For each TTP, we then return the hyper-parameters that achieved the highest F1-score for the given TTP on the validation set. While any other metric can be chosen, we pick F1-score in order to strike a balance between precision and recall (*i.e.*, true positive rate). To ensure the reproducibility and robustness of our results, we initialise the training of each of our decision trees with five different random states, and then subsequently proceed with validation and testing for each state. The random state parameter controls the randomness of the decision tree. We average out the results from each random state for every decision tree and report only the averaged values going forth. We also compute 95% confidence intervals for each metric, however we do not plot these intervals as the range of the standard deviations is quite small. Indeed, the maximum standard deviation we register across all policies and thresholds is equal to 1.1×10^{-4}, thus indicating the stability of our models.

3.3.2.2 Comparison of Different Aggregation Policies

The first analysis we conduct is a comparison of the different aggregation policies described in Section 3.2.3, considering 5 metrics: (*i*) accuracy, (*ii*) precision, (*iii*) true positive rate, (*iv*) false positive rate, and (*v*) F1-score. Each of these metric is necessary to better understand the capabilities of the system and to tune the policy parameters as necessary. For example, choosing a higher threshold will probably minimise the false positive rate, however, it is equally important to understand the impact this will have on the true positive rate. In Figure 3.6/left we plot the performance of each of the policies according to the listed metrics while varying the decision thresholds. As we can see from Figure 3.6/left/top, P1 has good performance for low θ_t (*i.e.*, $\theta_t = 1$ and $\theta_t = 2$), however, as expected, it soon becomes too strict and its true positive rate drops below 0.01 for $\theta_t = 3$. On the other hand, as we can see from Figure 3.6/left/middle and Figure 3.6/left/bottom, this does not happen for either P2 or P3. Further, we observe that all the policies start at almost the same values for our metrics (notice that P1 and P2 are equivalent for $\theta_t = \theta_f = 1$), but then different policies guarantee different levels of sensitivity when tuning their respective parameters, with P3 presenting the smoothest trend.

3.3.2.3 Comparison with Exclusively TTP-based Approach

The second analysis we conduct is to evaluate the need of the decision trees. To this end, we compare the results obtained by RADAR with Rule Based RADAR (RB-RADAR), which is RADAR without the TTP-based classification system. Exactly like RADAR, RB-RADAR is able to match each flow to the TTPs listed in Table 3.4. However, instead of using decision trees, RB-RADAR considers each flow matching at least one TTP as malicious. Then, in order to conclude

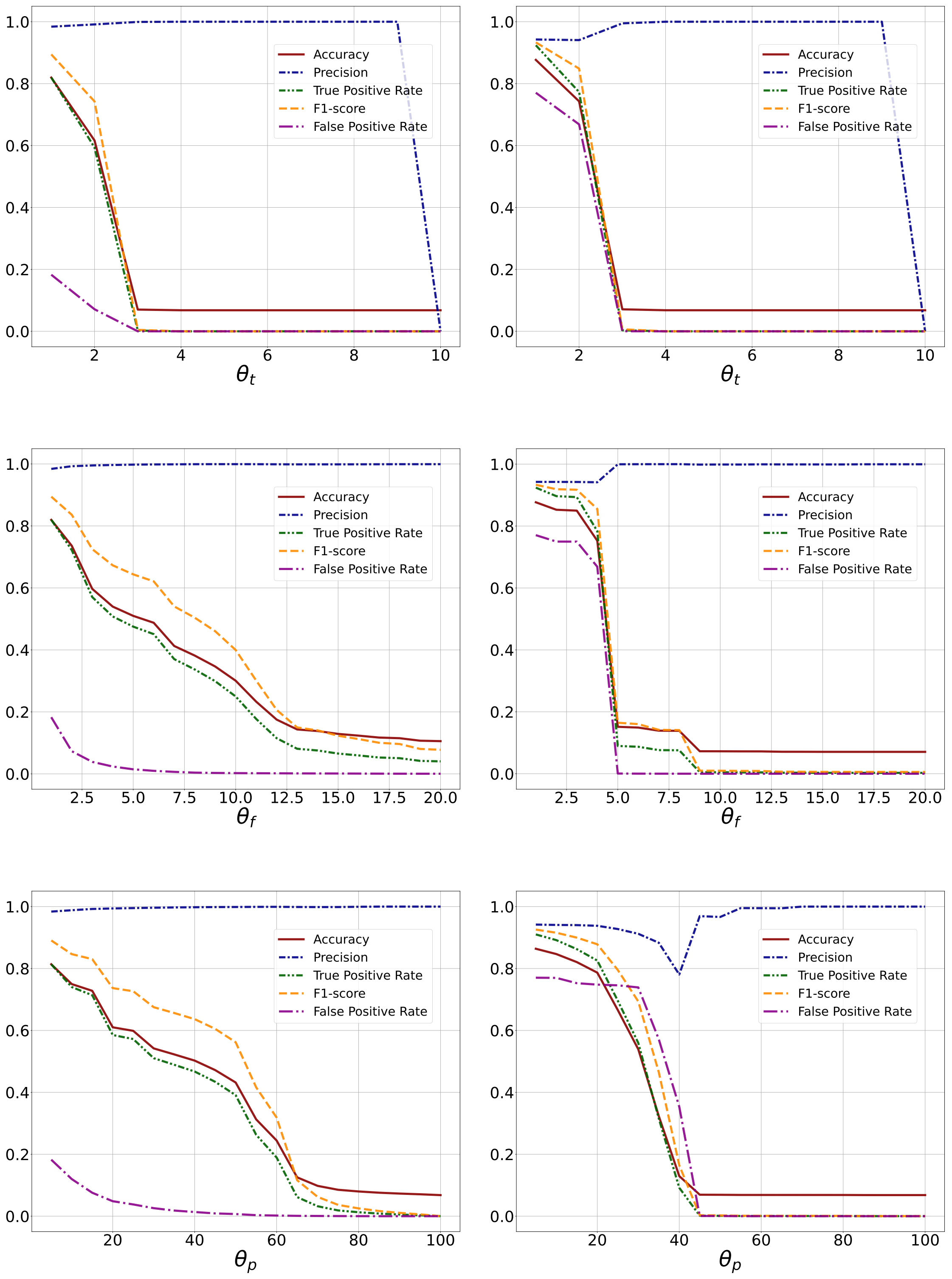


Figure 3.6: RADAR's (left figures) and RB-RADAR's (right figures) performances when using the 3 aggregation policies P1 (top), P2 (middle) and P3 (bottom) while varying the user defined threshold.

whether a sample is malicious or not, it uses the same 3 policies—P1, P2, P3—as RADAR. For this analysis we use all the same metrics as before, and we plot the results in Figure 3.6/right.

Firstly, we notice that for $\theta_t = \theta_f = \theta_p = 1$, RB-RADAR achieves not only high accuracy, precision, true positive rate, and F1-score, but also a very high false positive rate (nearly 0.8). In contrast, for the same thresholds, RADAR obtains a false positive rate below 0.2 while keeping the value associated to the other metrics high. Secondly, we observe that when RB-RADAR achieves a false positive rate less than 0.2 (with P1 for $\theta_t = 3$, with P2 for $\theta_f = 5$, and with P3 for $\theta_p = 45$), this results in a drastic reduction in accuracy, true positive rate and F1-score, which all become less than 0.2 as well (the precision, similar to RADAR, is very high no matter the chosen threshold). This shows that using only TTPs for malware classification is an unsuitable approach, and using a two-step classification system based on decision trees, is a far better strategy.

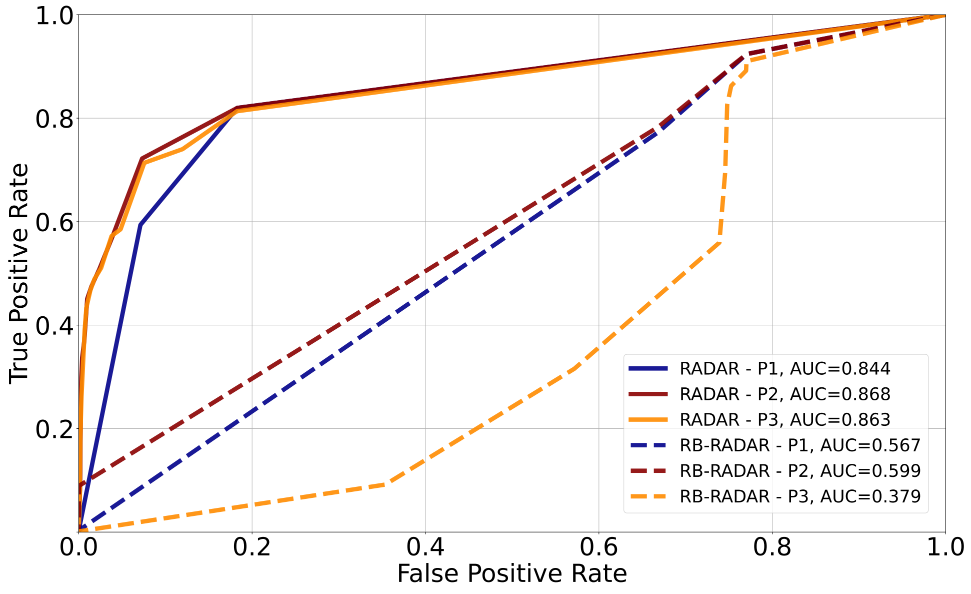


Figure 3.7: ROC Curves comparing RADAR and RB-RADAR.

Finally, in order to have a threshold independent comparison we plot the receiver operating characteristic (ROC) curve for each system and each of the three policies.

We further compute the area under the curve (AUC) for each of these ROC curves. The results of this comparison are plotted in Figure 3.7. As we can see from the figure, while RADAR is able to achieve an AUC always greater than 0.8, the performance of RB-RADAR is close to random classification when using P1 and P2, and worse than random classification when using P3. This further highlights that using decision trees to classify malware on the basis of TTPs improves performance while maintaining a balance between true positive and false positive rate.

3.4 Discussion

The development and testing of our system has interesting applications for malware analysis and the measurement of malware behaviour. In this section we discuss the results of our system development and testing.

Prevalence of Techniques. Based on our TTP extraction results, we find that only 8.62% of samples across all our datasets had no MITRE ATT&CK TTPs detected within them. While for just our malicious datasets, this number drops further to 7.56%. This suggests that TTPs are highly prevalent in malware and thus can be exploited for malware detection. The ATT&CK TTPs that we discovered to be prevalent within our datasets are also interesting. For instance, the use of non-standard ports to mask or blend-in malicious flows has been described by many threat intelligence reports and papers in isolated incidents and small datasets. However our research concludes that this is a common technique across the vast majority of malware in all of our datasets.

Variance of Techniques. Our results show that there is variance among the techniques used within each dataset. The frequency of these techniques is different since the underlying nature of the commodity malware datasets differs from one another. We find that some of these TTPs are more common across datasets, but also that the distribution of these TTPs is not uniform. This variation can be used to evaluate the TTPs that are currently being employed by malware authors and concentrate resources for mitigation accordingly. At the same time, the different usage of TTPs across benign and malicious samples, are strong indicators of malicious activity if the TTPs are primarily used by malware.

False Negatives. Previous work has shown that evasion of detection systems by changing features of network traffic has been a long-standing technique used by different malware families to evade network-based detection [73, 23]. We take steps to mitigate false negatives like this by detecting divergence from the standard port usage, as detailed in Section 3.2.2. This is a validation step not taken by comparable systems and a practical technique for threat intelligence analysts in a SOC.

Detection Sensitivity and Thresholds. During the course of system development, we found that selecting the appropriate detection sensitivity for various techniques was crucial. An improperly configured system which is not tuned to a specific environment could result in a high number of false positives. Network activity is not uniform or consistent across different environments, and requires tuning detection thresholds to match that environment. Our detection functionality offers configurable thresholds wherever possible, as demonstrated in Section 3.2.2. The experience and domain-knowledge of an analyst can be leveraged whilst utilising such a system in a SOC.

Opportunity for Improving Malware Detection. Our system is designed to integrate cohesively with existing IDS and EDR systems. The goal of our system is to improve detection of malicious behaviour by operating in conjunction with other systems. For instance, an IDS that utilises host-based analysis of malware can be paired with the network-based analysis of our system to more comprehensively analyse a given sample. Additionally, our system's focus on broader tactics and techniques make it useful across a broad spectrum of malware, the effectiveness of which can be increased by combining it with other NIDS that focus on the detection of specific malware families.

Decision Trees and Effectiveness. Our results highlight the importance of exploiting decision trees in our ML pipeline. We find that a system relying solely on TTPs can have a false positive rate almost as high as 0.8, if tuned to maximise other

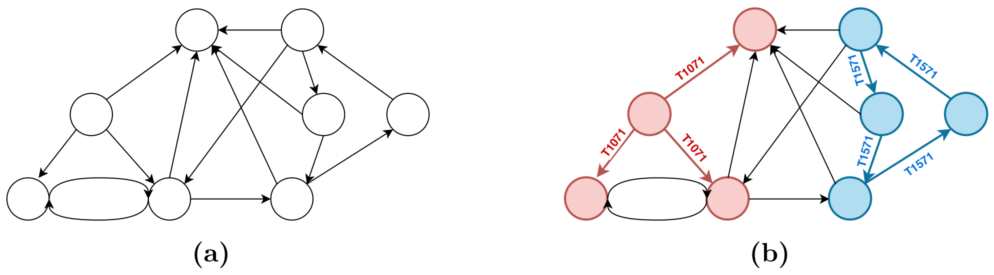


Figure 3.8: Network traffic graph views before (Figure (a)) and after (Figure (b)) the TTP Detection phase. The coloured nodes and edges represent hosts and flows respectively, which correspond to a detected TTP. In this example, the red flows represent the presence of *T1071 - Application Layer Protocol* while the blue flows represent *T1571 - Non-Standard Port*.

metrics such as true positive rate, precision, accuracy, and F1-score. On the other hand, using a classification pipeline with decision trees reduces the false positive rate to less than 0.2 in the worst case, where each of the other metrics is maximised. As discussed earlier, any classifier can be used for the classification of the malicious activity in RADAR. However, when we performed some experiments using other popular ML models (in particular, Adaptive Boosting, Logistic Regression, Deep Neural Networks, Random Forest, EXtreme Gradient Boosting, and Support Vector Machines), we found that there was no significant variation in the performance, and indeed resulted in a less transparent and thus less explainable system.

Extensibility and Explainability. RADAR is clearly an extensible, interpretable and explainable system thanks to its modular architecture, the exploitation of decision trees, the definition of clearly defined procedures in each stage of the decision process. Here, we stress the importance of our graph-based representation of the network and of exploiting TTPs for explainability. Indeed, whenever a sample is labelled as malicious, the relevant malicious TTPs detected can be visually represented by our system, allowing for further visual inspection by a SOC analyst. Figure 3.8a shows the state of the graph representing the network before the TTP Detection phase commences. Despite containing all the network traffic data, this graph is not particularly informative to a SOC analyst during analysis. However as Figure 3.8b shows, once the final phases of RADAR have completed, there is a lot more visual information for a SOC analyst to use for further investigation.

The flagged hosts and flows corresponding to a TTP detection can be clearly marked, coloured, and annotated with their respective TTP. An analyst can then immediately take some actions based on the information presented in the graph.

3.5 Related Work

In the network domain, work has mostly centred around the extraction of MITRE ATT&CK TTPs from network traffic. For instance, the network IDS Zeek [142] is used in [119] to extract MITRE ATT&CK TTPs. Their paper discusses methods to identify several TTPs and provides prototypical analysis scripts to test a few of these TTPs. BZAR [125] is another such example which has similar capabilities, as it enables the extraction of certain MITRE ATT&CK TTPs from Zeek log files. While RADAR also has a TTP Extraction phase, it goes significantly further than these systems. For instance, just comparing this phase alone, McPhee's scripts identify 3 techniques, while BZAR can identify 12 techniques and 8 sub-techniques across 9 tactics. RADAR on the other hand utilises a selection of 9 techniques and 6 sub-techniques suited to NetFlow traffic, that altogether encompass 11 tactics from the ATT&CK framework. Compared to BZAR where a majority of techniques (13/20) cover just 3 tactics, RADAR is thus more suitable to detecting a wider variety of malicious activity. Additionally, unlike BZAR where most TTPs (18/20) rely primarily on DCE-RPC protocol messages to enable detection, RADAR does not depend upon any one protocol for its detection capabilities and hence can handle more versatile network traffic. Furthermore, while BZAR only allows for the configuration of the 'epoch' threshold used to set the period over which it analyses data to detect TTPs, RADAR supports a higher degree of configurability by virtue of: (*i*) supporting thresholds for specific TTPs, (*ii*) allowing the selection of different detection policies for the overall system, and (*iii*) having policy-specific thresholds, which altogether enable a granular balance between the true-positive and the false-positive rate in the TTP detection phase. Lastly, we go even further by using these TTPs to effectively design an extensible system able to classify malware based on NetFlows. While doing so, our system explicitly separates malicious from benign usage of TTPs, and exploits interpretable models to provide an explainable classification of maliciousness to an analyst using it.

3.6 Limitations

There are limitations on the capabilities of our system. Given that our system is a network-based intrusion detection system, it will not be effective against malware that only executes locally and has no network footprint. We are also restricted to analysing the features we extract from our source data. Whilst it may be possible to have a more granular analysis by using all possible information captured within a PCAP, we have attempted to strike a balance between the granularity of our data and the privacy and efficiency of our system, as discussed in Section 3.2.1. Other limitations of general dynamic analysis systems also apply to us since we analyse samples whose network activity traces have been captured in dynamic analysis environments. The network traces captured are time-limited and may not necessarily represent the entire range of malware activity, and as such any analysis on them may not be exhaustive. We address this by selecting an execution cutoff-time parameter that allows for the majority of a sample's behaviour to be observed, as elaborated in Section 3.3. Additionally, malware may specifically evade detection systems by changing the features of its network traffic [149, 73, 23]. As we already pointed out, we mitigate this by detecting divergence from the standard port usage.

3.7 Conclusions

In this chapter we propose RADAR, a system that is capable of extracting MITRE ATT&CK TTPs from arbitrary network captures and using them to determine if the network traffic represents malicious behaviour. It is the first system that explicitly uses TTPs for malware detection, wherein the TTPs detected form the core of its detection pipeline and classification logic, rather than just serving as an additional part of the system. RADAR has been designed to be extensible and explainable. We demonstrate the effectiveness of RADAR by testing it against a dataset comprising of over 2 million samples, showing that RADAR is able to detect malware with an AUC score of 0.868, which is comparable to recent results in this domain [54]. Our experimental results also highlight that TTPs occur frequently in both malware and benign samples. As a result, TTPs alone are insufficient to detect malicious activity and decision trees are fundamental in achieving these results. To the best of our knowledge, RADAR is the first system that is able to effectively and automatically separate malicious usage of TTPs from benign usage, while also using interpretable models to provide an explainable classification of maliciousness to an analyst using the system.

It's amazing what you can do for love,
especially when it's unrequited.

— Edward Snowden

4 A Black-Box Approach for Exploiting TTPs for Malware Detection

4.1 Foreword

In Chapter 3 we focused on designing a specialised malware detection system that relied on existing human domain knowledge in the form of MITRE ATT&CK TTPs. Keeping in mind some of the shortcomings of existing IDSes, it was designed to be an open system—one that was extensible, explainable, and interpretable. In this chapter however, we flip the premise of the fundamental need and design of such an IDS: (*i*) what if our requirements did not require a specialised/new malware detection system, and there was an existing system in-place?, (*ii*) was there a way to exploit the benefits of using TTPs without redesigning existing security architecture?, and (*iii*) could such an approach be useful independent of whether explainable/interpretable or black-box machine learning algorithms were used in these IDSes , i.e., was it possible to design effective malware detection systems that exploited TTP information irrespective of the type of ML models being used? We tackle these questions and explore some of these challenges further in this chapter. We focus specifically on systems that utilise ML/AI algorithms for malware detection for two key reasons: (*i*) they are a crucial part of our methodology of effectively exploiting TTPs, by separating malicious and benign usage of TTPs as shown in the case of RADAR, and (*ii*) ML-based IDSes are heavily used in modern malware detection systems, and usually form a critical part of an organisation's security architecture. As such, answering these questions in the context of ML-based IDSes should prove highly beneficial to existing security architectures and help the improve the capabilities of future malware detection systems.

4.2 Introduction

Malware has been and continues to be a leading source of several problems within the context of cyber security. Be it data breaches, theft of sensitive data, or ransomware attacks, malware often plays a significant role in enabling these activities. For instance, according to the Malwarebytes 2022 Threat Review [113], 77% more malware has been detected in 2021 as compared to the previous year. Consequently, it is more important than ever to have robust malware detection and malware analysis capabilities. Several methods have been proposed by the security community over the past few decades to tackle some of these challenges. Among them, Machine Learning (ML) based Intrusion Detection Systems (IDS) have become an increasingly favoured technique for dealing with the rapid growth of malware, as a consequence of major advances in the field. The result is that it is now possible to tackle some of the key challenges in malware intrusion detection research, such as high false positive rates and adapting detection capabilities with constantly evolving malware behaviour [194]. While several ML-based IDSes have been proposed over the years, we find that these systems are usually purely data-driven, i.e., they only rely on the available data traces while ignoring the extensive domain knowledge that has been gathered along the years (see e.g., [19, 148, 61]).

In this chapter, we again focus on malware detection on networks and show how it is possible to leverage the available background knowledge in the form of the tactics, techniques, and procedures (TTPs) that are exhibited by malicious actors, and use them to enhance our ML-based system's malware detection capabilities. In particular, we use the TTPs from the industry-standard ontology for classifying the behaviour of malicious actors, namely the MITRE ATT&CK framework [126]. We use the same set of TTPs relevant for network traffic that were identified in the previous chapter in Section 3.2.2, and extract them in a similar manner from network traffic captures. However, contrary to the previous chapter our proposed approach reasons and operates on the sample-level (in contrast, RADAR operates at the flow-level), and as a result we design a novel representation for all the flows present in a given sample. This representation is additionally able to take TTP information as input, and include it in the form of TTP features. This single representation of the numerous flows exhibited by a sample, is called a Bag of Flows (BoF). This representation is a variation of the Bag of Words (BoW) model, that has been extensively used in the natural language processing field to create a vectorial representation of textual documents based on the frequency of word occurrences [17]. Having created BoF representations, we further annotate each sample with TTP-based features specific to the TTPs detected within the given sample. Finally, we take the BoF representation together with the TTP-based features and pass them as input to the preferred ML model(s) for malware detection.

In order to assess the effectiveness of our proposed approach, we consider 7 different machine learning models which are commonly used by both researchers and practitioners, and compare the performance of TTPxML models (i.e., the models that have been tested and trained using the TTP-based features) with the 7 standard ML models (i.e., the models that have been tested and trained without TTP-based features) on the exact same parameters—flow-based features, samples, dataset splits, ML hyperparameters—with the only differentiator being the TTP-based features. We conduct an extensive experimental analysis on 5 challenging datasets, consisting of a total of 1,551,535 malicious and benign samples, comprising of 37,981,348 flows. These datasets represent an optimal playground to demonstrate the utility of TTPxML models over the standard ML models because: (*i*) they consist of multiple datasets, each having completely disjoint test sets, which allows us to prove the robustness of our claims, (*ii*) they contain balanced target classes, which allows us to train the standard ML models under optimal conditions, and (*iii*) they contain a varied population of malware samples, which allows us to test our hypothesis on a challenging benchmark. Further, in order to make sure that the TTP-based features are actually helping ML models from every perspective, we used 4 different metrics to measure their performance: F1-score, Accuracy, True Positive Rate (TPR) and False Positive Rate (FPR). Our experimental analysis demonstrates that:

1. The TTPxML models consistently outperform the standard ML models. Indeed, when averaging over the 5 datasets, we compare the models a total of 952 times, and the TTPxML models outperform the standard models 922 times (i.e., 96.8% of the time).

2. The TTPxML models benefit particularly from the TTP-based features in data scarce regimes, i.e., when few training datapoints are available. Indeed, in such scenarios, we report an improvement of up to 3.7% in terms of F1-score, 2.3% in terms of accuracy, 7.66% in terms of TPR and 32.3% in terms of FPR.

3. The TTPxML models perform better than the standard ones when subjected to camouflaging attacks, i.e., when an attacker tries to camouflage malicious activity by deliberately generating large amounts of traffic from benign operations. Indeed, in such scenario, we report an improvement of up to 19.8% in terms of F1-score, 15.9% in terms of accuracy, 28.4% in terms of TPR and 84.9% in terms of FPR.

Thus, our experimental analysis provides strong evidence that the TTP-based features can substantially and robustly improve the performance of standard ML models.

To summarise, the key contributions of this chapter are:

1. We propose a methodology to exploit background knowledge about malware, in the form of TTP-based features, in order to improve the performance of ML-based intrusion detection systems. This methodology can be applied to

a wide-variety of machine learning methods, and can be easily extended to incorporate additional TTPs and/or ML models.

2. We evaluate the merits of this approach on 5 challenging datasets containing real-world malware samples and benign samples. By comparing the performance of our approach to various standard machine learning methods, we show that TTPxML models have superior performance independent of the ML model deployed, thus indicating the robustness of the proposed methodology. We further conduct a study on the size of the representation we propose, and find that our results hold true independent of any such variation.

3. We additionally test our hypothesis under challenging conditions, namely when the data is particularly scarce, or when the malicious samples have been camouflaged to resemble benign ones. Even in these scenarios, we find that our TTPxML models outperform the standard models.

The rest of the chapter is organised as follows. We describe our proposed solution and its methodology in Section 4.3. We then conduct a thorough evaluation of our approach in Section 4.4. Finally, we discuss the findings from our experiments in Section 4.5, and provide conclusions in Section 4.6.

4.3 Methodology

Our methodology consists of four basic steps:

1. We take as input the raw network captures exhibited by samples, and transform them into privacy-preserving Netflows.

2. We annotate each flow with all the TTPs that it matches.

3. For all flows corresponding to a particular sample, we then create a single Bag of Flows (BoF) which is a vectorial representation of the sample.

4. We then employ a variety of binary classifiers to decide whether a given sample is benign or malicious.

The following sections describe each of these steps in greater detail.

4.3.1 Network Flows

We use a similar procedure as described in Section 3.2.1 of Chapter 3 to ingest raw network traffic captures and transform them into NetFlows. For completeness, we provide a summary of the steps followed, below. Raw network traffic data is often captured and analysed as PCAP files using applications such as Wireshark [138] and tcpdump [87]. While this format has several advantages, the primary one being access to granular packet-level information, we instead opt to transform these network captures into NetFlows (or simply, flows). Flows are an aggregated summary of a network-level conversation between two hosts, typically containing metadata about the packets exchanged in the given conversation. While flows do not possess the same level of granular detail as PCAPs, they do have several advantages for our specific use-case: (*i*) they are lightweight in terms of processing time since they have a fixed length and format, (*ii*) their less granular compact data representation is much smaller in size and hence can be stored without a commensurate increase in storage costs, and (*iii*) they are privacy-preserving since they only store packet metadata and do not contain any actual data payloads. In particular, we use a specific type of NetFlow, Yet Another Flowmeter (YAF) [86], which is designed for high performance and scalability across large networks, while also balancing the need for capturing information and maintaining privacy on the network.

We thus transform the raw network captures collected into network flows. For each flow, we create a list of features that describe its properties, which are shown in Table 4.1. Before these flow properties are utilised further in our system, they undergo another step of processing (described in Section 4.3.3) that creates the final flow-level features that are used by the various ML models.

4.3.2 TTP Annotation of Flows

Out of all the techniques and sub-techniques described in the MITRE ATT&CK framework, we select the 15 which are exhibited by malware that utilise network

Field	**Property**
Flow Hash	The hash associated with each unique flow.
Sample Hash	Hash of the sample corresponding to the flow.
Unique ID	A unique ID associated with each flow.
Dataset	The dataset to which a flow belongs.
Source Port	The port from which the flow originates.
Destination Port	The port on which the flow connects to.
Start time, end time	Flow start or end time.
Duration	Flow duration in seconds.
Protocol	IP protocol identifier in decimal format.
Entropy	The Shannon Entropy for the flow payload.
Applabel	The application label, as identified by YAF.
Source IP, Dest. IP	Source and destination IPv4/IPv6 address.
Type, Code	ICMP type or code in decimal format.
Isn, Risn	Forward or reverse TCP sequence number.
Flags	4 properties representing various TCP flags.
Tag, Rtag	802.1q VLAN tag in forward/reverse direction.
Pkt, Rpkt	No. of packets in forward/reverse direction.
Oct, Roct	No. of bytes in forward/reverse direction.
RTT	Round-trip time estimate in milliseconds.
End-reason	Reason for termination of flow.

Table 4.1: Flow properties.

resources and can hence be detected in our datasets comprising of network flows. The selected techniques and sub-techniques can be found in Table 4.2. This is not meant to be a comprehensive list of all TTPs required for detecting network-based malware, but is instead comprised of a reasonably diverse set of tactics and techniques, such that the vast majority of network-based malware can be characterised using some combination of these TTPs. As a result, our chosen TTPs span a varied range of tactics used by malware, such as Reconnaissance, Credential Access, Discovery, Lateral Movement, Command and Control, and Execution.

In order to obtain TTP labels for our network datasets, we follow a similar approach to that proposed in [162] (and Chapter 3), and create a mapping between specific flow properties and TTPs. For instance, in order to detect the technique T1124 (System Time Discovery) we consider one specific procedure that is used for discovering time on the network, namely the Network Time Protocol (NTP) [124]. Since NTP

Tactics	**Techniques**
Reconnaissance	T1590 - Gathering Victim Network Information T1595 - Active Scanning
Credential Access	T1557.001 - Man-in-the-Middle
Discovery	T1046 - Network Service Scanning T1124 - System Time Discovery T1135 - Network Share Discovery
Lateral Movement	T1021.001 - Remote Services (RDP) T1021.004 - Remote Services (SSH) T1550.003 - Use Alternate Authentication Material T1563.001 - Remote Service Session Hijacking (RDP) T1563.002 - Remote Service Session Hijacking (SSH) T1570 - Lateral Tool Transfer
Command and Control	T1071 - Application Layer Protocol T1090 - Proxy T1105 - Ingress Tool Transfer T1571 - Non-Standard Port
Execution	T1053 - Scheduled Task/Job

Table 4.2: List of supported TTPs from the MITRE ATT&CK framework.

uses a specific port, 123, to send and receive time synchronisation messages, we are able to map any TCP or UDP flows that communicate using that port to the technique T1124.

4.3.3 Bag of Flows

In order to create the BoF representation of a given sample, we first need to create a vectorial representation of each flow. In order to do so, we need to transform each non-numerical feature to a numerical one. To this end, we consider the features “End-reason”, “Protocol” and the four features representing the TCP flags, which are all categorical, and we transform them into one-hot encoded vectors. The features “Isn” and “Risn”, though categorical, have too many unique values (63,796 and 8,229,092, respectively) and thus we drop them. We then consider the feature “Destination IP”, for which we want to retain the ordinal information inherent of IP addresses. Hence, we map each IP address into a 32-bit integer obtained in the following manner: given an IP address $w.x.y.z$, we map it to the number $(w \times 256^3) + (x \times 256^2) + (y \times 256^1) + (z \times 256^0)$. Having pre-processed the relevant

features, we drop the features “Flow Hash”, “Sample Hash” and “Unique ID” as they are simply identifiers of the flows and/or samples, together with the feature “Dataset” as it is a spurious indicator of the maliciousness of each sample. Finally, we drop the features “Source IP”, “Start Time” and “End Time” as our dataset is made of software executed in a sandbox, and thus, while these features might be useful in practice, they carry no useful information in our dataset.
Once the above steps have concluded, we get a vectorial representation of each flow which comprises of the flow-level features and the TTPs corresponding to each flow. Since, we are interested in classifying whether a sample is malicious or not, we now need to aggregate the available per-flow information.

Many different methods have been proposed in the literature for modelling data in such a manner. We opt for a variation of the Bag of Words model [88], which we call Bag of Flows (BoF). Bag of Words is a standard yet powerful way to represent documents as vectors in natural language processing, and it represents each document as an unordered set of words with their position ignored, keeping only their frequency in the document. In essence, given the vocabulary of known words and a document to be represented, each word in the vocabulary is associated to a feature, and the value of each feature is given by the frequency of occurrence of the associated word in the document.

While we propose our own version of BoF, there are other methods that are also called BoF in the literature. For example, Divakaran et al. [52] define a BoF as a set of flows localised in time, and sharing the same: (*i*) destination IP address, (*ii*) destination port, and (*iii*) protocol. They then classify each flow and finally reach a decision regarding the maliciousness of the BoF by majority voting. Zhang et al. [204] use the term BoF in a similar way, and once a BoF has been identified, a Naive Bayes model is used as a base classifier for each flow, and different aggregation policies (e.g., sum rule, the max rule, and majority voting rule) are tested to reach a conclusion about the maliciousness of the BoF. Their definition of BoF is thus closer to our definition of a network sample. On the other hand, similar to our approach, Bartos et al. [20] use the term BoF to indicate a vectorial representation of a network sample. Contrary to our approach however, Bartos et al. make the assumption that each sample must have at least 5 flows (to be able to compute a meaningful representation) and a maximum of 100 (to ensure that the computational cost is controlled and does not exceed predefined limits), and this is reflected in their implementation of the BoF. On the other hand, the minimum number of flows per sample in our datasets is 1 and the maximum is 65,576 (with an average of 23.9). We thus needed to create a BoF that is able to represent all various kinds of samples in our dataset.

To outline our procedure to create the BoF, let D^{cont} denote the number of continuous flow-level features, D^{cat} denote the number of categorical flow-level features, and T the number of TTPs. Given a sample, we perform the following steps:

- We pre-process each flow-level feature by applying min-max scaling. As a result, the maximum value of any feature in the sample is going to be 1, while the minimum value is going to be 0.

- We transform each flow-level feature in a normalised histogram (i.e., a histogram recording the proportion of cases that fall into each category). Since not all the features are categorical, for each continuous feature we define B bins, and then for each bin b_i $(1 \leq i \leq B)$ we record the proportion of flows in the sample having value belonging to b_i. The vectorial representation of this histogram is thus in $\mathbb{R}^{B \times D^{cont} + H}$, where $H = \sum_{j=1}^{D^{cat}} V_j$, V_j being the number of different values of the jth categorical feature.

- We take each TTP and we create two aggregate features: (i) one representing the proportion of flows in the sample that match the given TTP, and (ii) the other representing the total number of flows matching the given TTP (this feature is then normalised over all the samples). The vectorial representation of the TTP-based features is thus in $\mathbb{R}^{2 \times T}$.

- We finally gather some global features about all the TTPs, namely: (i) the number of unique TTPs matched by the sample, (ii) the total number of

Dataset	**Samples**	**Flows**
Ember Malicious [13]	435,741	26,567,527
MalRec [160]	37,763	12,273,502
MalShare [150]	1,268,923	32,310,907
VirusShare [183]	595,098	12,217,493
Ember Benign [13]	155,432	1,423,023
Total (unique)	2,131,832	83,371,480

Table 4.3: Datasets statistics.

TTPs matched by the sample, (*iii*) the number of flows matching at least one TTP in the sample, (*iv*) the proportion of flows matching at least one TTP in the sample, (*v*) the mean of the number of TTPs matched by the sample, and (*vi*) the standard deviation of the number of TTPs matched by the sample. All these features are then normalised over all the samples. The vectorial representation of these global features is thus in $\mathbb{R}^6$.

- Finally, we concatenate the three representations obtained above and construct a single vector in $\mathbb{R}^{(B \times D^{cont})+H+(2\times T)+6}$.

This final vector represents our BoF for a given sample, and is subsequently passed on to a binary classifier for determining whether that sample is malicious or not.

4.3.4 Classification

Once we have constructed the BoF for each sample, we then use a binary classifier to classify whether it is malicious or benign. In order to test our hypothesis that these TTPs help improve the performance of binary classifiers in detecting malware, we test a number of different machine learning models that are commonly used to detect malware on networks, as shown in Table 2.1. In particular, we test the following models:

1. Adaptive Boosting (AdaBoost) [65], is an ensemble method whose output is given by the weighted combinations of the outputs of a weak learner (in our case a decision tree with max depth set equal to 1). Its name is due to

the fact that at each iteration it re-assigns the weight to each instance, with higher weights assigned to incorrectly classified instances. For example, it was used in [92] to detect malware from URLs.

2. Decision Tree (DT) [151], a non-parametric model for binary classification. In our case, we train it to maximise the Gini gain of each split. It is often appreciated for its interpretability. For example, it was used in [188] to detect malware and classify its family, and by us in [162] to create an interpretable system for malware detection.

3. Logistic Regression (LR) [47], models the probability of an event taking place by having the log-odds for the event be a linear combination of the features. Given its popularity, an entire survey was dedicated to the application of this model to the problem of malware detection [8].

4. Deep Neural Network (DNN) [117], a neural network, which in our case, has been implemented as a feed-forward neural network with ReLU non-linearity and four hidden layers. Deep neural networks have been increasingly applied to the task of malware detection with increasingly positive results (see e.g., [45, 148]).

5. Random Forest (RF) [80], is an ensemble learning method which builds multiple decision trees at training time and then outputs the class chosen by the majority of the decision trees. Random forests are, for example, used in [169] to detect infected websites from the content of their pages and in [26] to detect botnets on networks.

6. EXtreme Gradient Boosting (XGBoost) [36] is the newest ensemble method tested here. It was developed to reduce the overfitting problem typical of random forest and AdaBoost, while also increasing the training speed. It was, for example, used in [25] for anomaly detection using both network-level and host-level features.

7. Support Vector Machine (SVM) [46] is one of the most robust machine learning methods for binary classification and its objective is to find the best hyperplane dividing the two classes. Due to the high number of samples in our dataset, we use a linear kernel. SVMs were used for malware classification on networks in [20], and in [83] to detect phishing attacks.

4.4 Evaluation

In this section we report the results of our extensive experimental analysis. This section is structured as follows, we first start by describing the datasets and metrics used to evaluate the proposed approach. We then conduct a detailed comparison between the TTPxML models and the standard ML models, and also compute the statistical significance of our results. Then, we test the hypotheses that the improvements obtained with the TTP-based features are robust with respect to: (i) variations in the amount of training data, (ii) variations in the bin size of the BoFs, and (iii) injection of benign flows into malware samples, i.e., camouflage attacks.

4.4.1 Dataset

The goal of our evaluation is to determine whether the TTP-based features are actually helping the various kinds of ML models to discern between malicious and benign samples. In order to effectively achieve our goal, firstly we would like to have multiple datasets, which would allow us to test the robustness of our claims. Secondly, we would like our datasets to contain a varied population of malware samples, so that we can test the helpfulness of the TTP-based features across a broad range of scenarios. Finally, we would like our datasets to be balanced; this way, the ML models can be trained under ideal conditions.

We thus gather network traffic data from five different datasets, four of which contain malware samples while only one, namely “Ember Benign”, contains benign traffic. The basic characteristics of each of these datasets are given in Table 4.3. As can be seen from the Table, most of the samples in our datasets are malicious. In order to meet the conditions we set in the previous paragraph, we decide to create

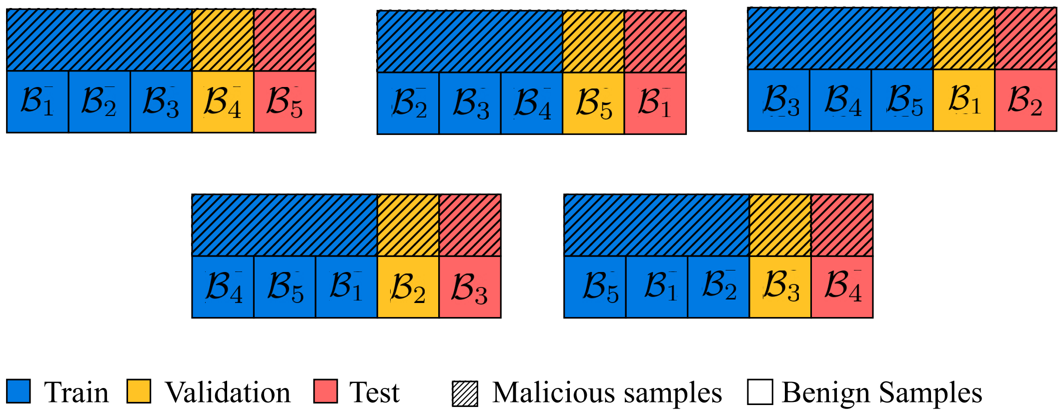

$\mathcal{B}_1$ $\mathcal{B}_2$ $\mathcal{B}_3$ $\mathcal{B}_4$ $\mathcal{B}_5$
$\mathcal{B}_2$ $\mathcal{B}_3$ $\mathcal{B}_4$ $\mathcal{B}_5$ $\mathcal{B}_1$
$\mathcal{B}_3$ $\mathcal{B}_4$ $\mathcal{B}_5$ $\mathcal{B}_1$ $\mathcal{B}_2$
$\mathcal{B}_4$ $\mathcal{B}_5$ $\mathcal{B}_1$ $\mathcal{B}_2$ $\mathcal{B}_3$
$\mathcal{B}_5$ $\mathcal{B}_1$ $\mathcal{B}_2$ $\mathcal{B}_3$ $\mathcal{B}_4$
Train
Validation
Test
Malicious samples
Benign Samples

Figure 4.1: Visualisation of the 5 newly-created datasets.

5 different datasets based on the datasets mentioned above, such that (*i*) they have balanced target classes, (*ii*) the test sets are disjoint, (*iii*) the malicious samples are randomly sampled from the four malicious datasets described in Table 4.3. To this end, we first create a single dataset of malicious samples $\mathcal{M}$ given by the union of the four datasets in Table 4.3. Then, we split the dataset of benign samples $\mathcal{B}$ (which is equal to Ember Benign) into 5 partitions $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ and $\mathcal{B}_4$. We create the ith dataset $(0 \leq i \leq 4)$ by performing the following steps:

1. we randomly sample without repetition from $\mathcal{M}$ a number of samples equal to the number of samples in $\mathcal{B}$,
2. we split the obtained malicious samples into train, validation and test,
3. we add the partitions $\mathcal{B}_{i\%5}, \mathcal{B}_{(i+1)\%5}, \mathcal{B}_{(i+2)\%5}$ to the train set, the partition $\mathcal{B}_{(i+3)\%5}$ to the validation set and the partition $\mathcal{B}_{(i+4)\%5}$ to the test set.

A visual representation of the final datasets is given in Figure 4.1. Notice that, given the size of $\mathcal{B}$, this procedure allows us to obtain 5 datasets with the highest number of samples such that they are balanced and their test sets are disjoint.

4.4.2 Metrics

In order to give a well-rounded overview of the performance of both types of models (TTPxML and standard ML) we decided to report their performance according to different metrics. In particular we chose: accuracy, F1-score, False Positive Rate (FPR) and True Positive Rate (TPR). Further, in order to test the statistical significance of our results, for each pair of models we performed the pairwise t-test.

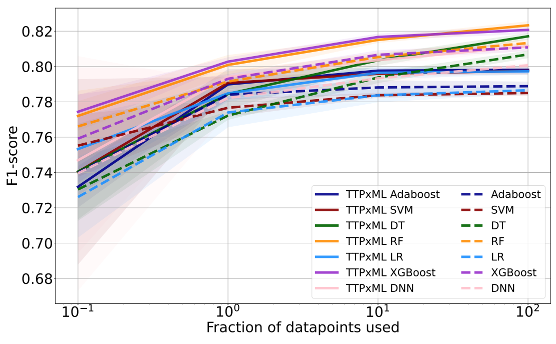
4.4.3 Detailed Comparison and Statistical Significance

We run all the models discussed in Section 4.3.4 on the five datasets generated as described in Section 4.4.1. In order to give a detailed comparison, we fix the bin size to 5, however, we will see later that the size of the bins does not impact our results. The average of the results (together with their standard deviation) according to the metrics listed above are given in Table 4.4.

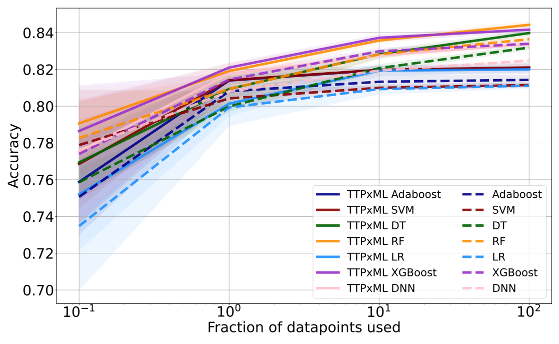
Let us look at the results according to each metric. If we focus on both Accuracy and F1-score, we can see that adding the TTP-based features always results in an improvement in performance. This empirically confirms that adding the TTP-based features can improve the performance of a model. All such improvements are also statistically significant (with p-value lower than 0.01) and can go to up 1.65% in terms of F1-score and 1.11% in terms of accuracy. In order to better understand how the TTP-based features are actually helping, we now focus on the FPR and TPR. As we can see from the results, the TTPs generally help in terms of reducing the FPR (with only two statistically not significant exceptions), however such results are statistically significant only for three models, with two models achieving p-value lower than 0.05 and one lower than 0.01. This tells us that the TTP-based features do not help (or rather, moderately help) the models to separate actually malicious samples from benign samples that "look" malicious. However, if we look at the TPR, we notice that the TTPxML models always achieve statistically significant better performance than their standard ML counterparts with p-value always less than 0.01, with improvements of up to 1.99%. This clearly shows that the TTP-based features help in improving the "coverage" of the given models, i.e., thanks to the information provided by the TTP-based features, we are able to teach the machine learning models to recognise a wider range of malware samples.

	F1-score (↑)		**Accuracy** (↑)		**FPR** (↓)		**TPR** (↑)	
	Standard ML	TTPxML	Standard ML	TTPxML	Standard ML	TTPxML	Standard ML	TTPxML
AdaBoost	0.789 ± 0.002	**0.798 ± 0.002****	0.814 ± 0.002	**0.821 ± 0.002****	**0.065 ± 0.005**	0.067 ± 0.007	0.694 ± 0.002	**0.708 ± 0.004****
Decision Tree	0.807 ± 0.001	**0.817 ± 0.001****	0.832 ± 0.001	**0.840 ± 0.001****	0.038 ± 0.003	**0.036 ± 0.001**	0.702 ± 0.003	**0.716 ± 0.001****
Log. Reg.	0.787 ± 0.001	**0.797 ± 0.002****	0.811 ± 0.001	**0.820 ± 0.001****	0.074 ± 0.001	**0.068 ± 0.002****	0.696 ± 0.001	**0.708 ± 0.003****
DNN	0.802 ± 0.004	**0.811 ± 0.002****	0.827 ± 0.004	**0.834 ± 0.001****	**0.045 ± 0.006**	0.047 ± 0.004	0.700 ± 0.002	**0.715 ± 0.004****
Rand. For.	0.813 ± 0.001	**0.823 ± 0.001****	0.836 ± 0.001	**0.844 ± 0.001****	0.039 ± 0.002	**0.037 ± 0.001**	0.712 ± 0.003	**0.726 ± 0.002****
XGBoost	0.811 ± 0.001	**0.821 ± 0.001****	0.834 ± 0.001	**0.842 ± 0.001****	0.044 ± 0.002	**0.041 ± 0.001***	0.712 ± 0.003	**0.725 ± 0.001****
SVM	0.785 ± 0.001	**0.798 ± 0.001****	0.812 ± 0.001	**0.821 ± 0.001****	0.065 ± 0.001	**0.063 ± 0.002***	0.688 ± 0.001	**0.705 ± 0.001****

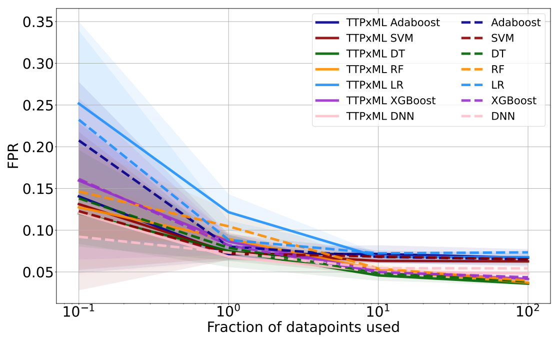
Table 4.4: Average and standard deviation of the results obtained by running each of the binary classification models on the five balanced datasets. For each metric, the ↑ (resp. ↓) indicates that the higher (resp. lower) the result, the better the performance. For each pair of results, the best is in bold. * (resp. **) indicates that the difference between the results is statistically significant with p-value < 0.05 (resp. < 0.01).



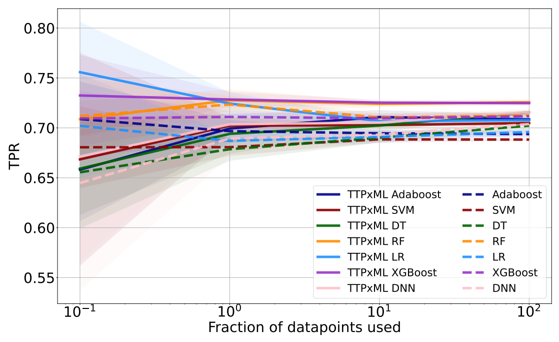
(a) F1-score



(b) Accuracy



(c) FPR



(d) TPR

Figure 4.2: Performance when varying the size of the training set. On the x-axis the percentage of randomly sampled datapoints from the training set ($\{0.1\%, 1\%, 10\%, 100\%\}$) can be found.

4.4.4 Robustness Under Data Scarcity

One of the limitations of machine learning systems is that they are often data greedy. We argue that TTP-based features can be especially helpful in assisting ML-based IDSes to detect malware in data scarce situations. This feature is particularly desirable because very few datapoints are likely to be available for novel malware, whenever it is discovered. This poses a significant challenge to IDSes who have to attempt to classify a type of malware they have not encountered frequently before. In order to test this hypothesis, we take the five datasets generated described in Section 4.4.1, and for each of them we generate three new training sets with 0.1%, 1%, and 10% of the datapoints, respectively. These datasets are obtained by randomly sampling over the original datasets and have sizes that range from 186 to 186182 datapoints.

We then train our models on the newly generated datasets, and plot the average of the results (obtained on the original full test sets) together with the results obtained with 100% of the training data in Figure 4.2. We also plot the 95% confidence interval for each result, which is represented by the shading around each line in the figure. As expected, the confidence intervals become smaller as the dataset size grows. If we focus on the averages, we notice that for most metrics and training dataset sizes, adding the TTPs results in an improvement of the average performance. Indeed, out of the 112 comparisons between the average results (4 percentages × 7 models × 4 metrics) we find that the TTPxML models outperform the standard ones 96 times (i.e., 85.7% of the time). The 16 cases where the standard models do better are all statistically not significant, as the paired t-tests for these results do not return a p-value < 0.05, with the exception of one (SVM at percentage 0.1 for metric TPR, where the t-test returns p-value 0.02). Further, in these 16 cases, the difference in performance is always negligible, with the only exception given by the performance of the logistic regression model, as judged by the FPR, which favours the standard model when training with 0.1% and 1% of the training dataset. This though is compensated by a significant improvement in performance according to the TPR, where we see that the model trained with TTP-based features shows an improvement of 7.66% over the standard model when training with 0.1% of the datapoints, and by 5.44% when training with 1%. Further, if we look at the results according to the F1-score and accuracy, we find that:

1. in terms of F1-score, the TTP-based model outperforms the standard one by 3.74% and 1.40% on the 0.1% and 1% datasets, respectively, and

2. in terms of accuracy, the TTP-based model outperforms the standard one by 2.35% and 0.26% on the 0.1% and 1% datasets, respectively.

Most importantly, from the charts in Figure 4.2 we can deduce that the TTP-based features are particularly helpful in data scarce situations, since the difference in performance between the TTPxML models and the standard ones normally increases as the training dataset size decreases. Indeed, when measuring the maximum difference in performance between the TTPxML models and the standard ones across all the metrics, we observe that:

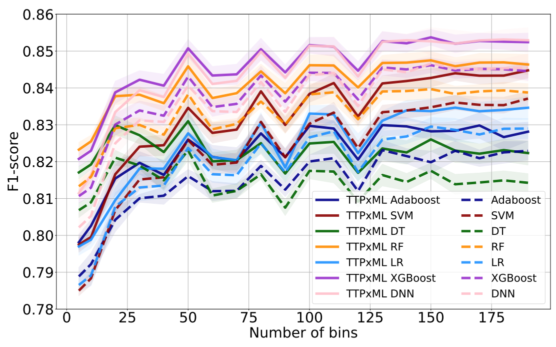
- when training with 100% of the datapoints, then the maximum improvement is 2.41% on the TPR, for the SVM model,
- when training with 10% of the datapoints, then the maximum improvement is 2.33% on the TPR, for the AdaBoost model,
- when training with 1% of the datapoints, then the maximum improvement is 5.44% on the TPR, for the logistic regression model, and
- when training with 0.1% of the datapoints, then the maximum improvement is 7.66% on the TPR, for the logistic regression model.

The fact that we consistently observe the maximum improvement in terms of the TPR, further reinforces the conclusions from the results shown in Table 4.4—namely, that the TTP-based features help improve the coverage of overall malware that can be detected by an ML-based IDS.

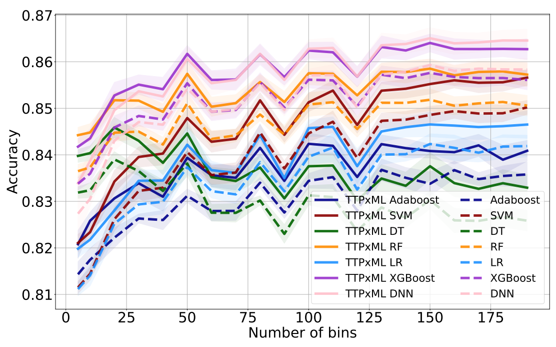
4.4.5 Robustness to Bin Size Variation

We now check whether the described improvements depend on the size of the chosen BoF representation. To this end, for each of the 5 balanced datasets, we vary the number of bins used to create the vectorial representation of each sample from 10 to 200 with step 10. We train each model on the BoF representations obtained with the different bin sizes, and plot the average performance over the 5 datasets in Fig. 4.3 (together with the performance for bin size 5, which is the size used in all our prior experiments). Again, we represent the 95% confidence interval with the shading around each average line.

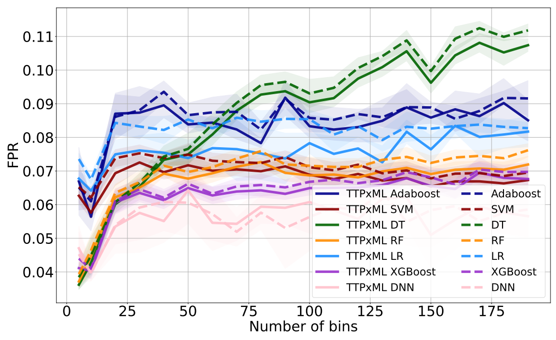
Consider the results plotted in Figure 4.3. If we focus on Figures 4.3a, 4.3b, and 4.3d we observe that the performance of all the models tends to increase as the bin size increases. In particular, we have a steeper rise from bins of size 5 to 25, and then from 30 onward the performance either stabilises or grows more slowly.



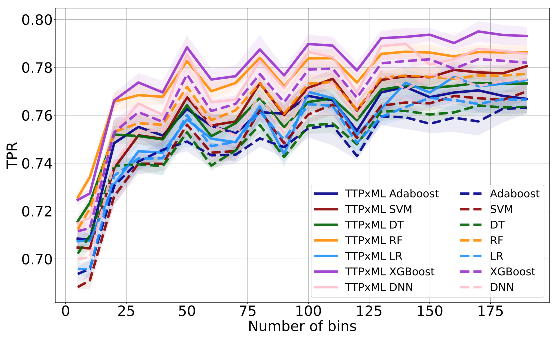
(a) F1-score



(b) Accuracy



(c) FPR



(d) TPR

Figure 4.3: Performance when varying the size of the bins to create the BoF. On the x-axis the size of the bins can be found.

However, if we look at Figure 4.3c we see the opposite trend: for most models we get the best (i.e., the lowest) FPR with bin size equal to 5, which then steeply increases between 5 and 25, and finally stabilises from bin size 30 onward for all models (with the exception of the decision trees for which the FPR keeps growing). The low FPR—together with the lower training and testing time for all ML models—explains why we chose to conduct all our previous experiments with bin size equal to 5.

Given the general trend followed by all the models, we now focus on the difference in performance between the TTPxML models and the standard ones. If we look at each metric, we see that:

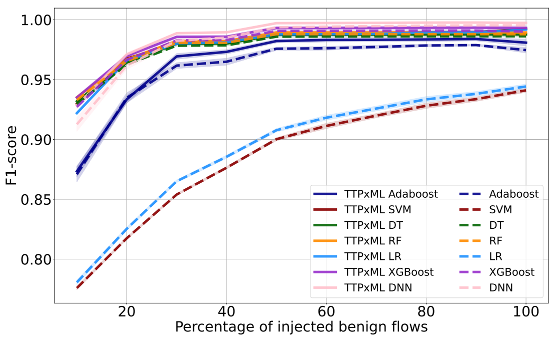
- in terms of F1-score, all the TTPxML models perform better than their standard counterparts for all bin sizes,
- in terms of accuracy, all the TTPxML models perform better than their standard counterparts for all bin sizes,

- in terms of FPR, all the TTPxML models perform better than their standard counterparts for all bin sizes with the exception of 5 instances, and
- in terms of TPR, all the TTPxML models perform better than their standard counterparts for all bin sizes with the exception of 4 instances.

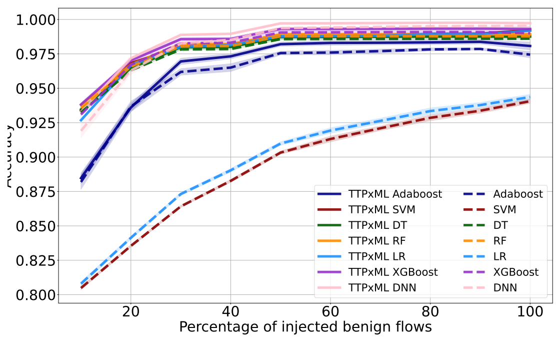
This is particularly impressive, as it highlights that out of all the 588 comparisons (7 models $\times$ 12 bin sizes $\times$ 4 metrics), the standard models manage to achieve better results than the TTPxML models only 9 times (i.e., the TTPxML models outperform them 98.4% of the time). Further, since in all of these 9 cases the difference in performance is again negligible (minimum difference equal to 8.4×10^{-5} and maximum difference equal to 0.006), we again perform a paired t-test to check the statistical significance of the results. As expected, all the p-values were > 0.5 but one (for the TPR metric with bin size 90 and model logistic regression for which the p-value is ~ 0.03), and thus all such results were not statistically significant with a single exception.

4.4.6 Robustness to Injection of Benign Flows

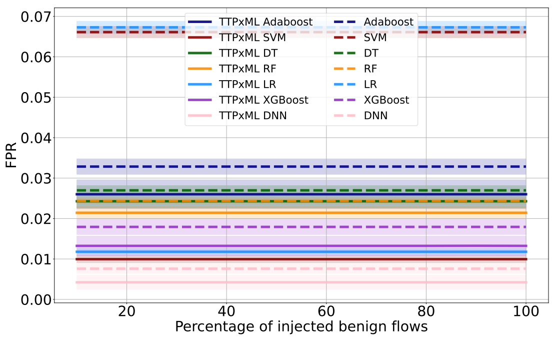
We consider an adversarial scenario, where the adversary attempts to evade detection by camouflaging their malicious activity using benign activities. To simulate this situation, we follow the approach proposed in [148] and inject varying amounts of benign flows into our malicious samples. Thus, for each of the 5 balanced training datasets, we obtain a novel training set by injecting different percentages of benign flows in the malicious samples. In particular, we consider percentage rates from 10% (i.e., given a malicious sample with N flows, we add $\lfloor \frac{N}{10} \rfloor$ random benign flows) to 100%, with step size 10, thus obtaining 10 different test sets for each of the 5 initial test sets. Our choice has two advantages: (*i*) it simulates a realistic scenario where we do not know what percentage of benign flow injection a malicious actor might use, and (*ii*) it allows us to evaluate whether the model is truly learning to detect malware traffic patterns, rather then potentially overfitting on the distribution of the specific benign dataset used.



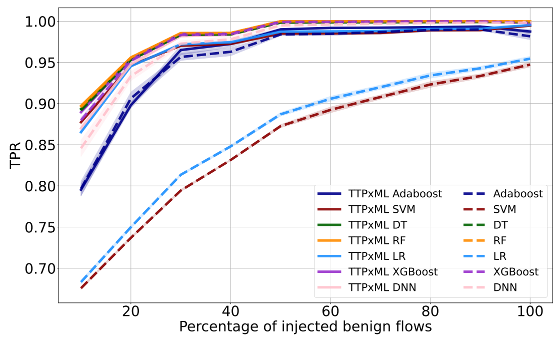
(a) F1-score



(b) Accuracy



(c) FPR



(d) TPR

Figure 4.4: Performance when injecting different percentages of benign flows in the samples at testing time. On the x-axis the percentage of injected benign flow can be found.

We plot the average performance (together with the 95% confidence interval) of our models when varying the levels of benign injection in Fig. 4.4. We immediately notice that: (*i*) the performance of all the models in terms of F1-score, accuracy and TPR increases as the percentage of benign flows injected in the malicious sample increases, and (*ii*) the FPR remains constant throughout the experiment. Regarding the former, it might seem counter-intuitive to observe that the task becomes easier as more benign flows are injected in the malicious samples. However, upon inspection, we notice that the average number of flows for malicious samples (with no benign flows injection) is 5 times higher than the average number of flows of the benign samples. Thus, injecting benign flows into the malicious samples makes the difference between malicious and benign samples even more striking. Notice that our models are able to pick up such trend due to our BoF representation, which provides a holistic view of each sample. On the contrary, in works such as

[148], camouflaging has a very negative impact on the performance of the model, as there the model only has access to fixed length sequences of flows. Regarding our second observation, the stable FPR is expected since the amount of benign data in each test set is constant.

Let us now focus on the comparison between the TTPxML models and the standard ML ones. As is evident from Figure 4.4, the TTP-based features are also helpful in this scenario. Indeed, we find that out of the 252 comparisons of the average performance (7 models × 9 percentages × 4 metrics) between the two types of models, the TTPxML models outperform the standard ML ones 247 times (i.e., 98% of the time). Further, in the 5 cases where it is not so, the difference in performance is negligible (minimum difference equal to 0.0001 and maximum difference equal to 0.008). We also perform a paired t-test to check the statistical significance of the results, and as expected, none of them return a p-value < 0.05, meaning that such differences are not statistically significant. On the contrary, we can see that for some models the introduction of TTP-based features results in a significant improvement of performance. In particular,

1. for the logistic regression, the TTPxML model's improvement over the standard model reaches a maximum of (*i*) 18.2% in terms of F1-score, (*ii*) 14.9% in terms of accuracy, (*iii*) 26.1% in terms of TPR, and (*iv*) 82.4% in terms of FPR,

2. for the SVM, the TTPxML model's improvement over the standard model reaches a maximum of (*i*) 19.8% in terms of F1-score, (*ii*) 15.9% in terms of accuracy, (*iii*) 28.4% in terms of TPR and (*iv*) 84.9% in terms of FPR.

Regarding the other models, the improvements—though consistent—are much more modest, as highlighted in Figure 4.4.

4.4.7 Overall Comparison

Overall, TTPxML models consistently perform better than standard ML models: when averaging over the 5 datasets, we compare the models a total of 952 times, and the TTPxML models outperform the standard models 922 times (i.e., 96.8% of the time). Of the 30 cases in which TTPxML has worse performance, 29 of those cases are not not statistically significant.

Focusing on F1-score, which is the standard metric used for comparison in the field, the TTPxML models outperform the standard ML models 235 times out of a total of 238 instances, i.e., our models have a higher F1-score 98.7% of the time. This consistent trend holds even in the best-case scenario for the standard models—100% datapoints, no injection, and minimum bin size—albeit with smaller gains. Overall, we find that regardless of the amount of data, the size of the BoF representation, adversarial conditions, type of ML model, or metric chosen; the TTPxML models consistently outperform the standard models.

4.5 Discussion

Coverage of Malware. Our results show that the key factor responsible for the overall increase in performance of TTPxML models is the improvement in TPR. This implies that using the TTP-based features, leads to more malicious samples being accurately identified as malicious than before. This follows from the fact that TTPs represent high-level adversarial behaviours, and thus by incorporating this knowledge in ML models, their ability to recognise such behaviour increases commensurately.

High Performance with Less Data. We also find that our approach is particularly useful in improving the performance of a model under data-scarce regimens. In particular, TTPxML models yield F1-scores greater or equal than those of standard models, even when trained with a tenth of the data used by the standard models. This has significant practical implications for cyber defence, given that malware is valued and feared as a function of its novelty—the less we know about it, or the less we have encountered it, the higher the risk it potentially poses (e.g., zero-day attacks).

Minimal Additional Data Cost. One of the most significant advantages of our approach is the fact that all this performance improvement comes at near-zero additional data cost. As discussed earlier, most ML-based systems are data greedy and require increasing amounts of data to improve performance. To the contrary, our TTP-based features are created from existing source data. The only additional data required is the knowledge contained within malware ontologies such as the ATT&CK framework, which are easily available, limited in size, and easy to store—none of which can be said about additional source data.

4.6 Conclusions

In this chapter, we propose a methodology for exploiting the domain knowledge of known adversarial behaviour, in the form of TTPs from the MITRE ATT&CK framework, for the purposes of improving ML-based malware detection on the network. Our approach automatically constructs TTP-based features from network traffic and provides this novel network sample representation as input to various ML models. We demonstrate the utility of our approach by evaluating our TTPxML models against standard ML models over 5 challenging datasets comprising of over 1.5 million samples and 37 million flows. Our results conclusively show that exploiting TTPs in this manner leads to better performance—indeed, models trained with the TTP-based features outperform the standard models across every metric, 96.8% of the time. Furthermore, TTPxML models: (*i*) have significantly better performance when less data is available, (*ii*) have similar or better performance than standard models while using a tenth of the data, and (*iii*) are robust in the face of adversarial attack.

Better to fail spectacularly than do something mediocre.

— Randy Pausch

5 Precise Malware Detection using TTPs

5.1 Foreword

In previous chapters we focused on designing malware detection systems that had a wide-variety of use-cases and requirements—they were either specialised or general-purpose, explainable and interpretable to a SOC analyst relying on their classification decisions or black-boxes that could improve the capabilities of any ML-based NIDS. Fundamentally though, all these systems were designed to exploit TTPs for malware detection in the broadest possible manner, a property that is desirable for any NIDS since it prioritises the malicious behaviours exhibited by the most frequently-occurring TTPs (and hence the vast majority of malware samples). Thus, TTPs that are not common in our malware datasets are effectively not prioritised in the best case and ignored in the worst case, when it comes to making detection decisions. This is especially true of data-driven methodologies, such as the various ML algorithms utilised in our systems. In this chapter, we model the task of network-based malware detection as a multi-label classification problem, a choice that enables us to: (*i*) detect malware more precisely on the basis of individual TTPs, and (*ii*) identify the malicious usage of uncommon or rarely-used TTPs. Aside from increasing the detection capabilities of a system, this approach opens up several new avenues for system optimisation and more detailed analysis of malware. We further generalise this approach and show that the ML-based malware detection systems described in Chapter 4, are a special case of this method and can be applied more broadly. For the purposes of our analysis, we choose a specific machine learning model that had previously proven effective in our experiments (namely DNNs), to design, compare, and evaluate our proposed approach.

5.2 Introduction

Machine learning methods and especially neural networks are now routinely used for malware detection in network traffic (see, e.g., [112, 164, 105]). Just within the research literature, Scopus indicates that there has been a 508% increase in papers referencing machine learning (ML) or deep learning (DL) within the context of cyber security over the last 5 years [159]. The rapid increase in the usage of such techniques within the domain of malware detection has been fuelled by the challenge of keeping up with constantly-evolving malware and the ability of ML models to handle large volumes of data. Approaches using neural networks in particular have been favoured since they do not require an expert's domain knowledge to define discriminative feature used in malware detection system [69]. Though very effective, such methods are often: (*i*) purely data-driven, ignoring the substantial body of available knowledge about the tactics, techniques, and procedures (TTPs) possibly used, and (*ii*) not precise, since they either cannot correlate malicious activity with TTP usage or if they do, they give little to no explanation about which TTPs have been maliciously used. Furthermore, ML methods typically require large amounts of quality training data, and are often not robust to adversarial attacks [69, 164].

In this chapter, we leverage the ontology of adversarial behaviour provided by the MITRE ATT&CK framework, comprising of tactics, techniques, and procedures (TTPs), to show that it is possible to precisely detect malware from network captures by (*i*) formulating the problem as a multi-label classification problem (see, e.g., [171]) with one set of labels to classify the TTPs, and another label to classify the maliciousness of the entire network traffic sample (henceforth referred to as *sample*), and (*ii*) designing a Deep Neural Network (DNN) architecture for the multi-label classification problem, that can take into account information about the TTPs utilised by the sample in the form of TTP features. In particular, we propose a methodology in which (*i*) each sample is analysed in order to automatically extract the TTPs utilised by it, (*ii*) the network traffic features of the sample, together with the features about the matched TTPs, are then given as inputs to the DNN architecture, which is not only responsible for detecting the maliciousness of the sample, but also which TTPs are maliciously used. To the best of our knowledge, ours is the first system which is able to detect malicious network traffic and precisely characterise which TTPs have been maliciously used. This is a rather useful property since it enables our system to have different fine-grained outputs, as advocated in [32]. Compared to the existing purely data-driven approaches for malware detection (see, e.g., [105, 112]), we expect that by (*i*) providing the information about the TTPs used by a sample, and (*ii*) training the neural network to detect both the malicious activity as a whole and the specific TTPs that are maliciously used, our system is particularly well-suited for detecting malware that utilises uncommon or rarely-used TTPs—a scenario which is particularly challenging for any data-driven approach.

We evaluate our system by conducting an extensive experimental analysis involving more than 1.5 million malicious and benign samples, curated from 5 different publicly available datasets, thus allowing us to test our system on a broad range of real-world samples. We consider 5 micro-average metrics usually used to evaluate systems for multi-label classification problems (F_1-score, Subset Accuracy, Precision, Recall and False Positive Rate), together with the macro-average metrics corresponding to the F_1-score, Precision and Recall. These metrics are widely used (see, e.g., [147, 175, 62]), and each highlights a different aspect of our system's performance. For instance, we utilise the macro-average metrics to highlight the performance of our system at TTP classification since individual TTP classes are unbalanced in our data, and these metrics are suited for multi-label classification with unbalanced classes [162, 198]. On the other hand, we use micro-average metrics to highlight the overall performance of our system, specifically for sample classification, which is evaluated on balanced malicious and benign datasets. We further evaluate our approach across four possible configurations:

1. STTP2STTP: the system is given the sample and TTP features as input, and it has to detect both if the sample is malicious, and which TTPs are maliciously used by it.

2. STTP2S: the system is given the sample and TTP features as input, and it has to determine whether the sample is malicious or not.

3. S2STTP: the system is given just the sample as input, and it has to detect both if the sample is malicious, and which TTPs are maliciously used by it.

4. S2S: the system is given just the sample as input, and it has to determine whether the sample is malicious or not.

We test our approach in all configurations, across all of our datasets, to thoroughly evaluate its performance. We conduct further experimental analysis to test our approach in various challenging conditions, such as when faced with limited training data or dealing with adversarial attack. Our experimental analysis confirms that:

- The STTP2STTP, STTP2S and S2STTP models consistently outperform the standard S2S model.

- The STTP2STTP model has the best overall performance, and significantly better average performance in detecting rarely-occurring malicious TTPs, as witnessed by an average improvement in the Macro F_1-score, Macro Precision and Macro Recall of 37.55%, 11.82% and 46.05% respectively.

- It is possible to further improve the performance of STTP2STTP by finely tuning the decision threshold for each individual TTP. In our datasets, we measure average performance gains of 12.5%, 12.0%, and 47.6%, for the Macro F_1-score, Macro Precision, and Macro Recall respectively.

- STTP2STTP maintains its ability to better detect malicious TTPs than S2STTP even when considering challenging scenarios where models are trained with a limited number of samples, or when tested with noisy malicious samples flooded with benign traffic by adversaries attempting to camouflage their activities.

The better performance of STTP2STTP when compared to any other model (be it S2STTP, STTP2Sor S2S) corresponds to the possibility of training STTP2STTP with less data and still getting the same performance of the other model, thus at least partially overcoming the well-known data-greediness problem of ML models (see, e.g., [69]). For instance, STTP2STTP utilising 25% of the training data has a higher average F_1-score than S2STTP utilising 100% of the training data, and STTP2STTP when trained with 10% of the data has on average still a higher Recall than S2STTP trained with 100% of the data.
To recap, the main contributions of the chapter can be summarised as follows:

- We design a novel multi-label DNN architecture that can precisely detect network-based malware along with the specific TTPs utilised by it, by exploiting TTP information in the form of automatically-generated features.

- We outperform the state-of-the-art models, regardless of whether or not they exploit the same TTP features used by our system. Our approach is particularly beneficial in detecting the malicious usage of uncommon or rarely-used TTPs.

- Our system is highly configurable, allowing for TTP by TTP tuning, thus further increasing its performance.

- We test our approach on a large dataset of real-world malicious and benign samples. We utilise 5 balanced datasets to fairly test our system, consisting of over 1.5 million samples.

- We evaluate our system under challenging scenarios, such as when it has access to limited training data or when it is subjected to adversarial attack, and demonstrate that it consistently outperforms other systems even under these conditions.

The rest of the chapter is organised as follows. Section 5.3 discusses the overall architecture of the system and the functioning of its individual components. The four models STTP2STTP, S2STTP, STTP2S and S2S are then comparatively evaluated in Section 5.4. Section 5.5 contains further experimental analysis of our system, by testing it under various challenging conditions and tuning it on a TTP-by-TTP basis to further improve performance. A thorough discussion about the results presented is in Section 5.6, followed by a comparison of our approach with closely related work in Section 5.7. Finally, we present the overall conclusions of our work in Section 5.8.

5.3 System Architecture

At a high level, our system classifies a given sample in the following manner:

1. A raw network traffic sample is taken as input, and modelled into a vectorial representation of the network traffic data, known as Bag of Flows (BoF).

2. TTP information about the sample is automatically extracted, and incorporated into the Bag of Flows in the form of TTP features.

3. This Bag of Flows is given as input to our multi-label DNN architecture, which performs the malware classification.

4. As per the model configuration, our system gives as output: (*i*) a malicious or benign classification for the overall sample, or (*ii*) a multi-label classification of each maliciously-used TTP by the sample, in addition to the malicious or benign classification for the overall sample.

We broadly describe the specifics of each of these steps in the subsequent sections.

5.3.1 Constructing the Bag of Flows

The first two steps are largely based on the methodology of the TTPxML system described in [163] (and Chapter 4). Here, to make the chapter more self-contained, we briefly describe how the raw network traffic samples are converted into their Bag of Flow representation, as required by the next step in our malware detection system. See Section 4.3.3 for more details.

Each raw network traffic sample is represented as a set of network flows, using Yet Another Flowmeter (YAF) [86], and annotated with the set of features in Table 5.1. YAF has several advantages, including the fact that it has been designed for high performance. Then, each flow is annotated with the 13 TTPs from the MITRE ATT&CK framework as shown in Table 5.2. This TTP matching is done at the flow level, and utilises the methodology described in [162] (and Chapter 3). These 13 TTPs encompass 11 tactics, 9 techniques, and 6 sub-techniques, and thus represent a sufficiently wide variety of adversarial behaviour that can be exhibited by network-based malware. Each flow is then converted into a vector in which (*i*) each non-numerical feature is converted into a numerical one, (*ii*) the categorical features "End-reason", "Protocol" and the 4 features representing TCP flags, are represented as one-hot encoded vectors, while (*iii*) "Destination IP" is converted into a 32-bit integer maintaining the structure information of IP addresses. Any flow properties that are not useful for classification or would unintentionally bias it, are dropped. Finally, the remaining flow properties are aggregated in a Bag of Flows, which is a vectorial representation of the original sample. Each BoF is constructed by (*i*) min-max scaling each flow-level feature, and (*ii*) transforming each flow-level feature into a normalised histogram recording the proportion of values that fall within a certain range. In practice, for each non categorical feature $B = 5$ bins are defined, and for each bin, the proportion of flows in the sample having a value belonging to that bin is recorded.

In the STTP2S and STTP2STTP models, the Bag of Flows representation also includes the following information about the TTPs matched by the flows in the sample:

Field	**Property**
Flow Hash	The hash associated with each unique flow.
Sample Hash	Hash of the sample corresponding to the flow.
Unique ID	A unique ID associated with each flow.
Dataset	The dataset to which a flow belongs.
Source Port	The port from which the flow originates.
Destination Port	The port on which the flow connects to.
Start time, end time	Flow start or end time.
Duration	Flow duration in seconds.
Protocol	IP protocol identifier in decimal format.
Entropy	The Shannon Entropy for the flow payload.
Applabel	The application label, as identified by YAF.
Source IP, Dest. IP	Source and destination IPv4/IPv6 address.
Type, Code	ICMP type or code in decimal format.
Isn, Risn	Forward or reverse TCP sequence number.
Flags	4 properties representing various TCP flags.
Tag, Rtag	802.1q VLAN tag in forward/reverse direction.
Pkt, Rpkt	No. of packets in forward/reverse direction.
Oct, Roct	No. of bytes in forward/reverse direction.
RTT	Round-trip time estimate in milliseconds.
End-reason	Reason for termination of flow.

Table 5.1: Flow properties.

- For each TTP two aggregate features are created: one representing the proportion of flows in the sample that match the given TTP, and the other representing the total number of flows matching the given TTP. This last feature is normalised over all the samples.

- Then, the following information about the matched TTPs is also provided: (*i*) the number of unique TTPs matched by the sample, (*ii*) the total number of TTPs matched by the sample, (*iii*) the number of flows matching at least one TTP in the sample, (*iv*) the proportion of flows matching at least one TTP in the sample, (*v*) the mean of the number of TTPs matched by the sample, and (*vi*) the standard deviation of the number of TTPs matched by the sample. All these features are normalised over all the samples.

At this point, the sample has been successfully modelled as a Bag of Flows, and its corresponding TTP information has also been incorporated into this BoF. This

Tactics	Techniques
Reconnaissance	T1590 - Gathering Victim Network Information
Credential Access	T1557.001 - Man-in-the-Middle
Discovery	T1124 - System Time Discovery T1135 - Network Share Discovery
Lateral Movement	T1021.001 - Remote Services (RDP) T1021.004 - Remote Services (SSH) T1550.003 - Use Alternate Authentication Material T1563.001 - Remote Service Session Hijacking (RDP) T1563.002 - Remote Service Session Hijacking (SSH) T1570 - Lateral Tool Transfer
Command and Control	T1071 - Application Layer Protocol T1090 - Proxy T1105 - Ingress Tool Transfer T1571 - Non-Standard Port
Execution	T1053 - Scheduled Task/Job

Table 5.2: List of supported TTPs from the MITRE ATT&CK framework.

vectorial representation of the given sample is then passed on to the classifier.

5.3.2 Multi-label DNN Classification

The multi-label DNN architecture utilised by our system has an input layer of neurons, that has the same size as that of the vector representing the Bag of Flows, which was created in the previous step. The neural network is implemented as a fully-connected feed-forward neural network with ReLU non-linearity and four hidden layers. The output layer of the DNN contains 14 neurons, 13 corresponding to the outcome for the 13 TTPs supported by our system, and the final neuron corresponding to the maliciousness of the entire sample. Thus, this architecture allows us to operate in both multi-label and binary settings, enabling us to test our system in various configurations.

As previously stated, our system allows for 4 different configurations depending on whether:

1. TTP features are given as input along with the sample's BoF or not, to the classification system.
2. The output of binary classification of the sample, should also include information about which TTPs are maliciously used by the sample or not.

These 4 configurations are: STTP2STTP (TTP information in both input and output), S2STTP (just the sample information in input and TTP information as output), STTP2S (TTP information in input and just the sample maliciousness as output), and S2S (just the sample information in input and sample maliciousness as output).

All the four classification tasks can be modelled as multi-label classification problems, and transformed into binary classification problems, when dealing with configurations where the system only has to determine whether the overall sample is malicious or benign. In a multi-label classification model, an input vector x of values is mapped to an output binary vector in which each element represents a property of the input sample x. In contrast to binary classification, an input sample x can have multiple elements in the output vector mapped to 1, thus corresponding to x satisfying the properties represented by such output values. Formally, as defined in [71], a *multi-label classification* problem is defined as a pair $(\mathcal{A}, \mathcal{X})$, where $\mathcal{A}$ is a set of labels, while $\mathcal{X}$ is a finite set of pairs (x, y) where $x \in \mathbb{R}^N$ $(N \geq 1)$ is a datapoint/sample, and $y \subseteq \mathcal{A}$ is the ground truth of x, i.e., the set of labels associated with x. A *model* m for $\mathcal{P}$ is a function $m(\cdot, \cdot)$ which maps every label A and datapoint $x \in \mathbb{R}^N$ to $[0, 1]$. For every label A the function $m_A : \mathbb{R}^N \mapsto [0, 1]$ is defined by $x \mapsto m(A, x)$, for each datapoint $x \in \mathbb{R}^N$. Finally, a datapoint $x \in \mathbb{R}^N$ is predicted by m to have label A whenever $m_A(x)$ is greater than or equal to a user-defined threshold $\theta_A \in [0, 1]$.

Ideally, we would like to have one label per TTP that we are interested in detecting. In our case, this would correspond to having the set of labels $\mathcal{A}$ equal to the set of TTPs in Table 5.2; the set $\mathcal{X}$ to be the set of Bag of Flows corresponding to the samples, in which each Bag of Flows x is paired with the labels y corresponding to the TTPs in Table 5.2 which are maliciously used by the sample. Such a picture has two difficulties:

1. All available datasets of malware network traffic only have information regarding the maliciousness of the sample, and do not contain any information about which are the TTPs being used.
2. The list of TTPs in Table 5.2 is by no means exhaustive, meaning that there can certainly be malicious network traffic samples that do not match any of the TTPs supported by our system.

We overcome the first difficulty by *(i)* automatically detecting the TTPs in each sample as described in Section 5.3.1, and *(ii)* associating the set of labels corresponding to the detected TTPs, to the sample.

To overcome the second difficulty, we introduce an additional label called Mal-Sample whose ground truth is assumed to be 1 when the sample is malicious, even if it does not match any of the TTPs our system supports. Notice that even assuming that our system would support all known TTPs, such an additional label is still useful given the possibility of either an attack using a never seen before TTP, or the TTP detection component failing to recognise a TTP (be it malicious or benign) in the network traffic sample. However, the presence of the label Mal-Sample indicating sample maliciousness and of the other labels indicating malicious usage of TTPs raises the following possibilities:

1. The TTP outputs and the sample output are compatible with each other, meaning that either all the TTPs in the sample and the sample itself are classified as benign, or that some of the TTPs in the sample and the sample itself are classified as malicious. In this case, no issue arises and all the TTPs and the sample are considered according to their predictions.
2. The TTP outputs and the sample output are different because all the TTPs in the sample are labelled as benign and the sample as malicious. Such a case reflects the possibility that the model has detected a malicious sample even

though it does not match any of the supported TTPs. Given this, all the TTPs and the sample are considered according to their predictions, as in the first case.

3. The TTP outputs and the sample output are different, implying that some TTP is labelled as malicious while the sample is labelled as benign. Such a situation does not reflect any real scenario, and it is possible given the inherent fact that machine learning models in general, and neural networks in particular, may fail to learn existing relationships between the labels (see, e.g., [72], for a discussion on this). In such a case, taking a prudential approach, we also consider the sample as malicious and count it accordingly in the metrics.

Indeed, from a technical point of view, the problem we are considering is the special case of a multi-label classification problem with hierarchical constraints between the labels. Special techniques have been developed for such problems in order to guarantee coherency between hierarchical labels, (see, e.g., [191, 70, 158]). We did not consider such techniques given the relative simplicity of our hierarchy with only one level, in which, as we said, coherency is guaranteed by simply flipping the Mal-Sample label when necessary. In practice, such incoherence shows up very rarely: in our experiments there are no differences (up to the third decimal place) in the F_1-score, Precision, and Recall of label Mal-Sample when measured before and after Mal-Sample is turned to 1, in order to solve the incoherence between label Mal-Sample and the TTP labels.

5.4 Evaluation

The goal of our experimental evaluation is to assess the benefits of (*i*) providing TTP information as additional features to the neural network; and (*ii*) teaching the neural network to detect the malicious usage of TTPs, when classifying a given sample. Specifically, we pose the following questions:

1. Is it possible to effectively detect the malicious usage of each TTP?

Dataset	**Samples**	**Flows**
Ember Malicious [13]	435,741	26,567,527
MalRec [160]	37,763	12,273,502
MalShare [150]	1,268,923	32,310,907
VirusShare [183]	595,098	12,217,493
Ember Benign [13]	155,432	1,423,023
Total (unique)	2,131,832	83,371,480

Table 5.3: Statistics about the original datasets.

2. In the previous case, do the TTP features help?
3. Does detecting the malicious usage of TTPs, also help solve the simpler problem of determining the maliciousness of the overall sample?
4. In the previous case, do the TTP features help?

Towards this end, we comparatively evaluate our different models STTP2STTP, S2STTP, STTP2S and S2S on the same 5 datasets used in [163] (and Chapter 4). In the following subsections, we first provide a detailed description of the datasets, highlighting some of their characteristics. Then, we present the metrics used in our evaluation. Finally, we present the results of our experimental evaluation, addressing the first two questions in Subsection 5.4.3, and the third and fourth questions in Subsection 5.4.4.

5.4.1 Datasets

We use the same 5 balanced datasets we created in Chapter 4, to enable a fair comparison across the various approaches. However, we provide much greater detail about the breakdown of the datasets by individual TTPs in Table 5.4, since in this chapter we are stratifying the problem of malware detection by TTP. The datasets consist of a varied population of malware and benign samples from the five publicly available datasets represented in Table 5.3. Of these five datasets, only one, namely "Ember Benign", contains benign samples. Starting from the datasets in Table 5.3, five new datasets have been constructed by randomly sampling the five

	Train (tot. samples = 186183)				**Validation** (tot. samples = 62060)				**Test** (tot. samples = 62061)			
	Benign		**Malicious**		**Benign**		**Malicious**		**Benign**		**Malicious**	
	#s	%	#s	%	#s	%	#s	%	#s	%	#s	%
T1557	0.0	0.00%	0.4	0.00%	0.0	0.00%	0.0	0.00%	0.0	0.00%	0.2	0.00%
T1135	24.0	0.03%	362.2	0.39%	8.0	0.03%	114.2	0.37%	8.0	0.03%	119.6	0.39%
T1124	61962.6	66.56%	71892.6	77.23%	20654.2	66.56%	23935.8	77.14%	20654.2	66.56%	23976.2	77.27%
T1071	9.0	0.01%	46.8	0.05%	3.0	0.01%	14.2	0.05%	3.0	0.01%	14.8	0.05%
T1590	2.4	0.00%	0.2	0.00%	0.8	0.00%	0.0	0.00%	0.8	0.00%	0.0	0.00%
T1563	0.0	0.00%	0.0	0.00%	0.0	0.00%	0.0	0.00%	0.0	0.00%	0.0	0.00%
T1105	97.2	0.10%	42.4	0.05%	32.4	0.10%	12.4	0.04%	32.4	0.10%	11.4	0.04%
T1090	1.2	0.00%	44.6	0.05%	0.4	0.00%	10.4	0.03%	0.4	0.00%	15.0	0.05%
T1550	0.6	0.00%	10.8	0.01%	0.2	0.00%	3.6	0.01%	0.2	0.00%	3.6	0.01%
T1570	0.0	0.00%	0.0	0.00%	0.0	0.00%	0.0	0.00%	0.0	0.00%	0.0	0.00%
T1571	71584.2	76.90%	86088.8	92.48%	23861.4	76.90%	28652.6	92.34%	23861.4	76.90%	28683.8	92.44%
T1053	3.0	0.00%	0.8	0.00%	1.0	0.00%	0.2	0.00%	1.0	0.00%	0.0	0.00%
T1021	11.4	0.01%	28.6	0.03%	3.8	0.01%	8.4	0.03%	3.8	0.01%	8.6	0.03%
Tnull	21548.0	23.15%	6986.0	7.50%	7045.0	22.70%	2416.0	7.79%	7178.0	23.13%	2299.0	7.41%
Total	133695.6		158518.2		44565.2		52751.8		44565.2		52833.2	

Table 5.4: Statistics about the newly-derived datasets. Average number and percentage of matched samples per TTP, across the train/validation/test sets. Column #s is the number of matched samples. The Total number of TTPs does not include Tnull, since Tnull represents the samples that do not match any TTPs.

original ones. Each of the five new datasets has then been split into training (60% of the samples), validation (20% of the samples), and test (20% of the samples) sets. These new datasets share the following properties: (*i*) they are balanced at the sample level, meaning that they contain a balanced number of malicious and benign samples, (*ii*) they are split ensuring that they have pairwise disjoint test sets, (*iii*) they are as large as possible given the previous two requirements. Though these new datasets should share the same distributions of samples, by considering five (and not just a single dataset) we are able to compute the mean and standard deviation of our results, thus minimising the risk of skewed results due to particular distributions in the training, validation, or test sets.
Table 5.4 describes the mean statistics of our data across all five of our training, validation, and test sets. It also shows for each individual TTP:

- The number of malicious and benign samples that contain the TTP.
- The percentage of malicious and benign samples that contain the TTP.

When we look at the last row of Table 5.4 which shows the total number of samples that match a TTP (excluding samples that do not match any TTP), we observe that there is not a significant difference between the number of benign and malicious samples with TTPs (~15% difference). Further, notice that the average number of TTPs matched by benign samples is 1.4, while it is 1.7 for malicious ones. The second observation is that the dataset is highly unbalanced at the TTP level: T1124 and T1571 occur in most of the malicious and benign samples, while most of the others occur rarely in the datasets, with T1557, T1590, T1563, T1570, T1053 not occurring at least once on average in our train/validation/test sets. This implies that our models cannot always be properly trained/validated/tested on the labels corresponding to such TTPs as there are no samples with such labels in the corresponding train/validation/test sets. For this reason, we do not consider the labels associated with T1557, T1590, T1563, T1570, T1053, and only consider the 9 labels T1135, T1124, T1071, T1105, T1090, T1550, T1571, T1021, each associated to the corresponding TTP, plus the label Mal-Sample corresponding to the entire sample. Finally, while the most frequent TTPs often occur in malicious samples, the opposite happens for some other TTPs, like T1105 and T1053.

5.4.2 Metrics

Given the ith label A_i (representing either a TTP or the label Mal-Sample associated to the sample), we assume

1. P_i and N_i denote the sets of malicious and benign samples with label A_i (and thus $|P_i| + |N_i| = n$ where n is the number of samples), and
2. PP_i and PN_i denote the sets of predicted malicious and predicted benign samples respectively, with label A_i (and thus also $|PP_i| + |PN_i| = n$).

Then, all the metrics are defined on the basis of the *confusion matrix* represented by the different values for P_i, N_i, PP_i and NN_i of the different labels. Considering the ith label A_i,

1. the *True Positive set* TP_i is the set of samples which are predicted to be positive (malicious) and are actually positive (malicious):

$$TP_i = P_i \cap PP_i.$$

2. the *False Positive set* FP_i is the set of samples which are predicted to be positive (malicious) and are actually negative (benign):

$$FP_i = N_i \cap PP_i.$$

3. the *True Negative set* TN_i is the set of samples which are predicted to be negative (benign) and are actually negative (benign):

$$TN_i = N_i \cap PN_i.$$

4. the *False Negative set* FN_i is the set of samples which are predicted to be negative (benign) and are actually positive (malicious):

$$FN_i = P_i \cap PN_i.$$

On the basis of the confusion matrix of each label, many metrics have been defined and used to evaluate model performance in multi-label and binary classification problems. A standard way to present the different metrics is to distinguish between "micro" and "macro" average scores, the latter usually distinguished by appending the word "Macro" in front or after the name of the metric.

In the micro case, each metric groups the classifications by sample across all labels, and then calculates the overall metric, thus giving each prediction equal importance. This, hence favours the labels with high *support*, defined as the number of samples having a particular label. In contrast, each macro metric calculates the average of the metric across all classes, thus giving each label equal importance, irrespective of their support. In general, macro metrics are better suited in highlighting differences in performance when considering highly unbalanced datasets, as is our case when looking at the data at the TTP-level.

We use the following micro and macro scores to present our results. See, e.g., [147] for a more in depth presentation of such metrics, including a study about the correlations existing among them.

5.4.2.1 Micro-average metrics

If we define $TP = \sum_i |TP_i|$, $FP = \sum_i |FP_i|$, $TN = \sum_i |TN_i|$, $FN = \sum_i |FN_i|$, $P = \sum_i |P_i|$, $N = \sum_i |N_i|$ and $PP = \sum_i |PP_i|$ the following (micro-average) metrics are among the most commonly used:

1. *F_1-score (F_1)*, it is the harmonic mean of Precision and Recall, defined as:

$$F_1 = \frac{2PrRe}{Pr + Re} = \frac{2TP}{2TP + FP + FN} = \frac{2TP}{PP + P}.$$

2. *Accuracy (Acc)*, it is defined as the ratio between the number of correctly predicted labels to the total number of predictions:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{P + N}.$$

3. *Precision (Pr)*, it is the proportion of the correctly predicted positive labels to the overall positively predicted labels, i.e., the ratio between the correctly predicted positive labels and the total number of positively predicted labels:

$$Pr = \frac{TP}{TP + FP} = \frac{TP}{PP}.$$

4. *Recall (Re)*, it is the ratio between the correctly predicted positive labels and the total number of actually positive labels:

$$Re = \frac{TP}{TP + FN} = \frac{TP}{P}.$$

5. *False Positive Rate (FPR)*, it is the proportion of the incorrectly predicted positive labels to the number of actually negative labels:

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N}.$$

Another standard metric used in multi-label classification, defined directly on the basis of the ground truth y_i and the predicted labels $\hat{y}_i$ of each sample x_i, is Subset Accuracy. *Subset Accuracy (SA)* is defined as the percentage of samples that have all their labels classified correctly:

$$SA = \frac{1}{n}\sum_{i=1}^{n} 1(y_i = \hat{y}_i)$$

where n is the total number of samples and 1(.) is the indicator function returning 1 if the condition in parenthesis is true, and 0 otherwise.

5.4.2.2 Macro-average metrics

In multi-label classification problems, macro-average metrics are computed by averaging the corresponding micro-average metric as defined in Section 5.4.2.1. So, if N is the number of labels,
Macro F_1-score is defined as:

$$\textit{Macro } F_1\textit{-score} = \frac{1}{N}\sum_{i} \frac{2|TP_i|}{2|PP_i| + |P_i|},$$

	STTP2STTP	S2STTP
F_1-score (↑)	**0.835 ± 0.003**	0.826 ± 0.003
Accuracy (↑)	**0.972 ± 0.000**	0.970 ± 0.000
Subset Accuracy (↑)	**0.841 ± 0.002**	0.831 ± 0.002
Precision (↑)	**0.953 ± 0.002**	0.951 ± 0.002
Recall (↑)	**0.744 ± 0.005**	0.730 ± 0.005
FPR (↓)	**0.003 ± 0.002**	0.004 ± 0.000

Table 5.5: Average and standard deviation of the results obtained by running STTP2STTP and S2STTP on the five datasets, detecting the malicious TTPs. For each metric, the ↑ (resp. ↓) indicates that the higher (resp. lower) the result, the better the performance. For each pair of results, the best is in bold.

Macro Precision is defined as:

$$\textit{Macro Precision} = \frac{1}{N}\sum_i \frac{|TP_i|}{|PP_i|},$$

and *Macro Recall* is defined as:

$$\textit{Macro Recall} = \frac{1}{N}\sum_i \frac{|TP_i|}{|P_i|}.$$

In our case $N = 9$, corresponding to the 8 TTPs appearing on average at least once in our train/validation/test sets, plus the label Mal-Sample associated to the entire sample.

5.4.3 Evaluating Malicious Usage of TTPs

Our first experiment tests the effectiveness of detecting malicious usage of TTPs and assesses the extent to which providing TTP features is helpful to a system's overall malware detection capabilities. We thus consider our STTP2STTP and S2STTP models, where the only difference between them is whether TTP information is provided as input, but otherwise identical input data and configurations are used. Specifically, we train a feed forward neural network with 4 hidden layers (each with 1250 neurons) and the following hyper-parameters: dropout rate of 0.4, learning rate of 1.6×10^{-4}, and weight decay of 3.05×10^{-6}. All the models are trained using mini-batching (with batch size of 750) and Adam optimizer [95].

The results of this experiment are presented in Table 5.5, and the best results for each metric are highlighted in bold. Looking at the table, we observe that, firstly as expected, providing TTP features consistently helps across all metrics. Secondly, the absolute values are quite high for each metric, and in particular, it is remarkably high for Accuracy ($\simeq 0.97$, meaning that more than 97% of the predictions are correct) and Precision ($\simeq 0.95$, meaning that more than 95% of the positive predictions are correct). We obtain the "lowest" value for the Recall metric ($\simeq 0.74$, meaning that our models are able to detect roughly three out of four of the positive labels, each representing maliciousness).

Next, we see that though the TTP information helps, the difference between the values associated with each metric does not appear to be remarkably high if we consider the (micro) statistics in Table 5.5. This fact underlines that, assuming we are interested in maximising such micro-average scores, it is possible to not provide the TTP features as input and let the model learn how to detect and properly classify each TTP by itself, with a relatively low decay in the micro-average performance.

However, the true benefits of the STTP2STTP system over S2STTP can be seen when analysing the results of this experiment on a TTP-by-TTP basis. When considering our selected metrics at the individual TTP level, we observe significant improvements in performance when utilising the TTP features, as in the case of STTP2STTP.

A careful analysis of the Precision, Recall, and F_1-score of each label, reported in Table 5.6, reveals that:

1. TTP features are beneficial in all cases, except for T1021.
2. The contribution of the TTP features is relatively modest for the highly occurring TTPs (T1124 and T1571), and significantly higher for many of the others. For instance, if we consider T1105, the F_1-score for STTP2STTP is 0.686, and 0 for S2STTP.

Such facts are reflected by the macro-average scores reported in the last line of Table 5.6, where we see huge improvements across all metrics:

	STTP2STTP			S2STTP			
	F_1 (↑)	Precision (↑)	Recall (↑)	F_1 (↑)	Precision (↑)	Recall (↑)	Support
T1135	**0.962**	**0.986**	**0.940**	0.872	0.884	0.866	119.6
T1124	**0.850**	**0.952**	**0.768**	0.838	0.950	0.752	23976.2
T1071	**0.656**	**0.742**	**0.590**	0.566	0.806	0.458	14.8
T1105	**0.686**	**0.660**	**0.726**	0.000	0.000	0.000	11.4
T1090	**0.946**	**0.972**	**0.922**	0.078	0.600	0.044	15.0
T1550	**0.160**	**0.200**	**0.134**	0.000	0.000	0.000	3.6
T1571	**0.840**	**0.952**	**0.752**	0.830	0.954	0.736	28683.8
T1021	0.000	0.000	0.000	**0.310**	**0.594**	**0.236**	8.6
Mal-Sample	**0.820**	**0.952**	**0.718**	0.810	0.950	0.708	31031.0
Macro avg	**0.658**	**0.713**	**0.617**	0.478	0.638	0.422	9318.2

Table 5.6: Average of the scores corresponding to each label. For each metric, the ↑ (resp. ↓) indicates that the higher (resp. lower) the result, the better the performance. For each pair of results, the best is in bold. For each label, column Support reports the number of (malicious) samples with that label.

1. Macro F_1-score jumps from 0.478 to 0.658, with an increase of 37.55%.
2. Macro Precision goes from 0.638 to 0.713, with an increase of 11.82%.
3. Macro Recall jumps to 0.617 from 0.422, with the highest increase of 46.05%.

In Section 5.5.1 we will see how such positive results can further be improved by fine-tuning the threshold θ associated with each label to decide the maliciousness of the corresponding TTP and/or sample.

As a final consideration, we remark that all the statistics reported in Tables 5.5 and Table 5.6 take into account the metrics associated with all the labels, including the label Mal-Sample associated with the entire sample. This is standard practice in both multi-label classification problems and hierarchical multi-label classification problems (see, e.g., [182]). Further, if we do not count the outputs of Mal-Sample, the statistics in Table 5.5 and the macro-average statistics in Table 5.6 would only be moderately affected. Indeed, the Precision and Recall of the label Mal-Sample associated with the sample (in Table 5.6) are very close to the overall Precision and Recall in Table 5.5, and consequently the two F_1-scores of Mal-Sample computed by STTP2STTP and S2STTP (equal to $\simeq$ 0.82 and $\simeq$ 0.81, respectively) are also similar to the overall F_1-score in Table 5.5 (equal to $\simeq$ 0.83).

5.4.4 Evaluating Sample Detection

We now want to evaluate how the detection of TTPs and their classification as malicious has an impact on the classification of the entire sample as either malicious or benign, and the impact of providing TTP features. We implement STTP2S and S2S with the same configuration and hyper-parameters as STTP2STTP and S2STTP. The results obtained by running all the models (i.e., STTP2S and S2S as well) are reported in Table 5.7, where the best results are highlighted in bold, while the second best are underlined. The last line reports the average ranking, obtained by ranking the performance of each system on a metric and then averaging the results. In the case of ties between models, average ranks are assigned to such models (see [49] for more details).

	STTP2STTP	S2STTP	STTP2S	S2S
F_1-score (↑)	**0.820 ± 0.002**	<u>0.811 ± 0.003</u>	<u>0.811 ± 0.002</u>	0.802 ± 0.004
Accuracy (↑)	**0.842 ± 0.002**	<u>0.835 ± 0.002</u>	0.834 ± 0.001	0.827 ± 0.004
Precision (↑)	**0.952 ± 0.002**	<u>0.951 ± 0.002</u>	0.939 ± 0.005	0.949 ± 0.005
Recall (↑)	**0.720 ± 0.004**	0.708 ± 0.004	<u>0.715 ± 0.004</u>	0.700 ± 0.002
FPR (↓)	**0.037 ± 0.002**	**0.037 ± 0.002**	0.047 ± 0.004	0.045 ± 0.006
Avg ranking	1.1	2.2	3.1	3.6

Table 5.7: Average and standard deviation of the results obtained by running each of the four models on the five datasets, to detect malicious samples. For each metric, the ↑ (resp. ↓) indicates that the higher (resp. lower) the result, the better the performance. For each metric, the best result is in bold, and the second best is underlined. The last row reports the average ranking.

Considering the table, the first observation is again that STTP2STTP has the best performance across all the different metrics. Further, STTP2STTP almost always has the most stable performance of the lot, as measured by the standard deviation: the only exception to the rule is for Recall, for which STTP2STTP has a standard deviation of 0.004, equal to the standard deviation of S2STTP and STTP2S, but higher than that of S2S. Then, the second most performing model is S2STTP, which underlines the fact that the precise detection and classification of TTPs improves the overall performance of the corresponding binary classifiers. This result may look surprising given that (*i*) both STTP2STTP (resp. S2STTP) has been trained on the same dataset used for training STTP2S (resp. S2S), and (*ii*) STTP2STTP and S2STTP are actually solving a more complex problem than their corresponding models STTP2S and S2S, namely identifying malicious usage of TTPs and determining the maliciousness of the overall sample. However, by having a separate label for each TTP, the neural network has now the possibility to focus on the detection of the malicious samples containing rarely used TTPs, and classify them accordingly. The relatively good performance of S2STTP even when used as a binary classifier confirms that (*i*) it is possible to not provide TTP features as input, (*ii*) have the model learn how to detect the malicious usage of TTPs, and (*iii*) still not get a huge performance decay.

Additionally, model STTP2S has better performance than of S2S, confirming the fact that providing TTP information improves the model's performance, as also highlighted by the better performance of STTP2STTP compared to S2STTP.

Overall, the main results of this experimental analysis are that:

1. Teaching the neural network to detect the malicious usage of individual TTPs helps improve the overall performance of the system, as demonstrated by the performance metrics for STTP2STTP and S2STTP.

2. Using TTP features in this multi-label DNN architecture, greatly benefits the detection of the malicious usage of individual TTPs that occur rarely (i.e., have low support). As evidenced by the relative comparison of STTP2STTP to S2STTP at the TTP level in Table 5.6, performance across each metric increases significantly for nearly every such TTP.

3. Providing TTP features as input is a net benefit to any system and improves overall malware detection capabilities. This is demonstrated by the comparative performance of otherwise identical system, such as: STTP2STTP vs S2STTP, and STTP2S vs S2S.

4. Teaching the neural network to detect the malicious usage of TTPs and providing TTP features, leads to the best overall results, as demonstrated by STTP2STTP and its results across all metrics.

5.5 Experimental Analysis

In this Section, we present the results of further experimental analysis showing the results of (*i*) tuning the label decision thresholds in STTP2STTP (Section 5.5.1), (*ii*) training STTP2STTP and S2STTP with fewer samples (Section 5.5.2), and (*iii*) testing STTP2STTP and S2STTP with samples in which benign flows have been injected in the malicious samples in an attempt to camouflage their malicious activity (Section 5.5.3).

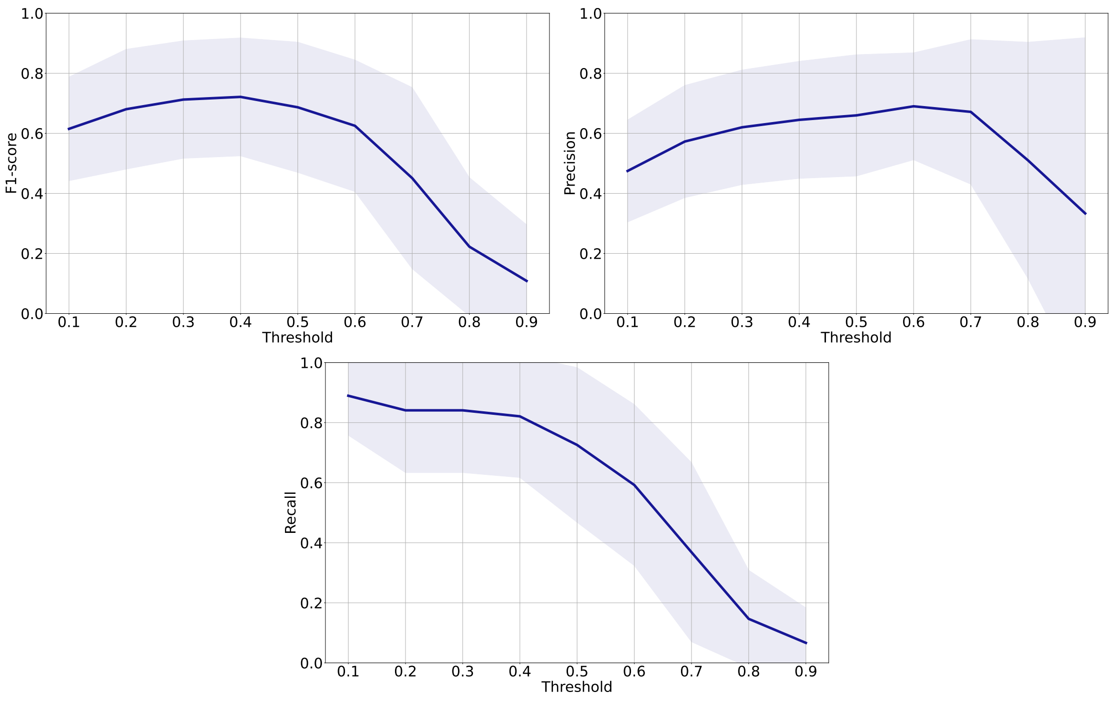


Figure 5.1: Average and confidence intervals of F_1-score, Precision and Recall for T1105 while varying the threshold θ of the TTP label.

5.5.1 Threshold Variation per TTP

The already positive performances of STTP2STTP reported in Section 5.4.3, can be further improved by tuning the decision threshold θ associated to each label and above which the corresponding TTP is predicted to have been maliciously used. For instance, if we consider TTP T1105 and plot its F_1-score, Precision and Recall scores while varying its threshold, we obtain the graphs in Figure 5.1. We observe that the F_1-score for T1105 peaks at some value in between $(0, 1)$ which is not necessarily equal to 0.5, which is the standard and intuitive threshold we used in our experiments. Indeed, we can expect F_1-scores to have a low value when the threshold is (extremely) low or (extremely) high. For instance, considering T1105, the average F_1-score across the five datasets is equal to $11.4/62061 \simeq 2\times10^{-4}$ when $\theta = 0$ (i.e., equal to the ratio between its average support, and the average number of samples in the test set), and equal to 0 when $\theta = 1$ (as in this case no sample is predicted to maliciously use T1105).

At the standard setting $\theta = 0.5$, the F_1-score for T1105 is 0.686, which raises to

0.721 (5.1% improvement) if θ is set to 0.4. A similar behaviour is observed for the Precision and Recall for T1105, which peak at 0.6 and 0.9 respectively. By selecting an appropriate value for θ for a given metric, we can further tune the system performance as per the desired requirement. The benefit of this approach is further highlighted by the fact that we can tune this threshold for each individual TTP, thereby allowing the system to incorporate the varying behaviours of each TTP while maximising its ability to better detect malicious activity. This can be seen for T1105 in Fig. 5.1, where the variation of the threshold θ is plotted for each metric along with its corresponding confidence interval.

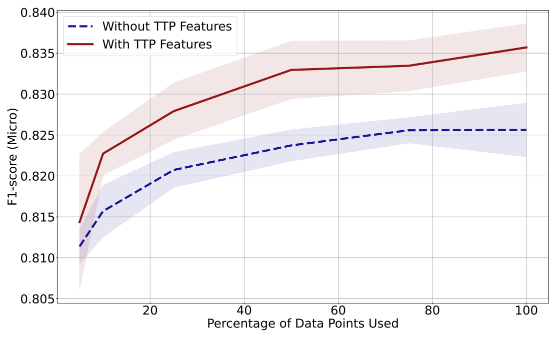
A comprehensive study of the impact of this feature is shown in Table 5.8, which reports the best result for Precision, Recall, and F_1-score for the various labels while varying their respective thresholds. We find that substantial improvements can be obtained across all metrics for each and every TTP. These improvements are highlighted by the macro-average metrics reported in the last line:

1. Macro F_1-score increases from 0.658 to 0.741, an improvement of 12.5%.
2. Macro Precision increases from 0.713 to 0.799, an improvement of 12.0%.
3. Macro Recall increases from 0.617 to 0.911, with the highest overall improvement of 47.6%.

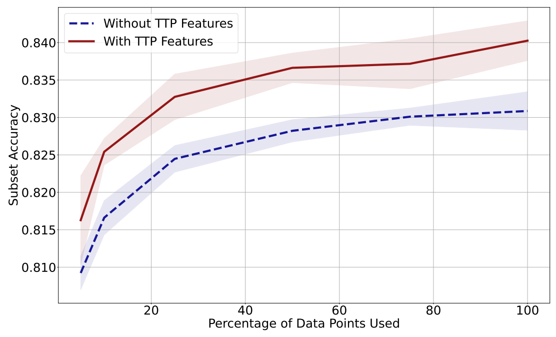
Of course, such fine-tuning at the TTP level is also possible for S2STTP, while it is not for either STTP2S or S2S.

5.5.2 Data Scarcity

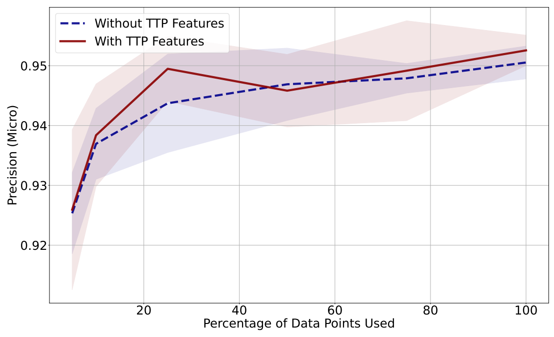
One of the known problems of machine learning models is that their performance is very sensitive to the amount of data used for training. As such we want to evaluate: (*i*) how robust our models are to the variation in training data provided to them, and (*ii*) what is the impact of the TTP features provided to them, when the amount of training data provided to the models is varied. Thus we design an experiment with two configurations of our system that allow such a comparison: STTP2STTP and



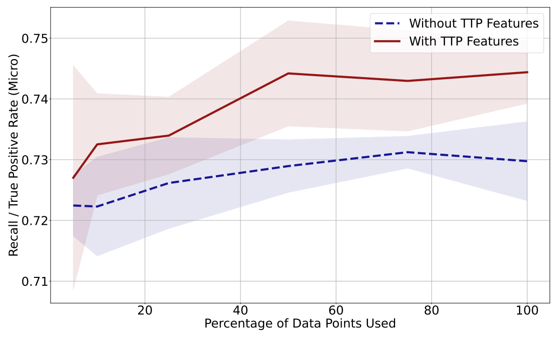
(a) F_1-score



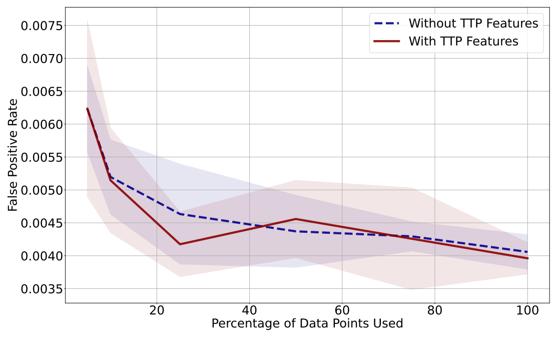
(b) Subset Accuracy



(c) Precision



(d) Recall



(e) FPR

Figure 5.2: Performance of the models when varying the size of the training set. On the x-axis the percentage of randomly sampled datapoints from the training set can be found.

	F_1 score					Precision					Recall				
	θ	value	θ_{max}	value	inc.	θ	value	θ_{max}	value	inc.	θ	value	θ_{max}	value	inc.
T1135:	0.5	0.963	0.4	**0.967**	0.4%	0.5	0.988	0.8	**0.994**	0.7%	0.5	0.940	0.1	**0.982**	4.4%
T1124:	0.5	0.850	0.4	**0.851**	0.1%	0.5	0.952	0.9	**0.979**	2.9%	0.5	0.768	0.1	**0.998**	30.1%
T1071:	0.5	0.656	0.3	**0.716**	9.2%	0.5	0.741	0.3	**0.891**	20.3%	0.5	0.591	0.1	**0.820**	38.7%
T1105:	0.5	0.687	0.4	**0.721**	5.0%	0.5	0.660	0.6	**0.690**	4.6%	0.5	0.726	0.1	**0.890**	22.6%
T1090:	0.5	0.947	0.6	**0.953**	0.7%	0.5	0.973	0.6	**0.987**	1.4%	0.5	0.922	0.1	**0.946**	2.6%
T1550:	0.5	0.160	0.1	**0.652**	307.7%	0.5	0.200	0.1	**0.610**	205.0%	0.5	0.133	0.1	**0.771**	478.6%
T1571:	0.5	0.841	0.4	**0.842**	0.2%	0.5	0.954	0.9	**0.980**	2.8%	0.5	0.752	0.1	**0.998**	32.8%
T1021:	0.5	0.000	0.1	**0.141**		0.5	0.000	0.1	**0.078**		0.5	0.000	0.1	**0.793**	
Mal-Sample	0.5	0.820	0.4	**0.821**	0.2%	0.5	0.952	0.9	**0.980**	2.9%	0.5	0.719	0.1	**0.998**	38.7%
Macro avg		0.658		**0.741**	12.5%		0.713		**0.799**	12.0%		0.617		**0.911**	47.6%

Table 5.8: Precision, Recall and F_1-score at different thresholds with the STTP2STTP model. θ_{max} is the threshold in between [0.1,0.9], step 0.1, for which we get the maximum resulting value. For $\theta = 0.5$ we report the same values of Table 5.5 in order to make comparison easier.

S2STTP. The idea is to vary the amount of training data given as input to STTP2STTP and S2STTP, while keeping all other variables equal, and measuring their relative performance to understand the impact of the TTP features in data scarce situations.

We expect the model STTP2STTP to be less dependent on the amount of training data used, thanks to the utilisation of the TTP information as model features. Thus, the existing gap between the performance obtained with STTP2STTP and S2STTP should either remain unvaried or increase as we train with less data. To test our hypothesis, we consider each of the five datasets used for training our models and randomly sample it in order to obtain 5 new datasets, each containing either 5%, 10%, 25%, 50% or 75% of its datapoints. Thus, for each percentage, we have 5 new sets of samples, that we use to train both STTP2STTP and S2STTP, which are then tested on the same test sets used in Section 5.4.
The results of our experiment are shown in Figure 5.2, where we also incorporate the performance of the models when trained with 100% of the samples. We also plot the 95% confidence interval for each result, which is represented by the shading around each line in the figure. Considering the figures, we see that:

1. As expected, STTP2STTP has better performance than S2STTP when considering F_1-score, Subset Accuracy and Recall.

2. The relative performance gap between the two models is not constant and varies with the amount of training data used. Generally, this gap increases with the amount of training data supplied.

3. While STTP2STTP has generally better performance, S2STTP does perform better when considering Precision and False Positive Rate, at some specific datapoints.

Such behaviour can be explained by considering the average characteristics of our five datasets reported in Table 5.4. Indeed, the TTPs with low support in the original datasets are likely to have even lower support in the new smaller datasets used for training, as the percentage of data used for sampling decreases. The availability of fewer datapoints, particularly for TTPs with already low support, is likely to correspond to a degradation in performance, as reflected in the plots in Fig. 5.2. At the same time, the micro-average metrics shown in Fig. 5.2 that favour the highly supported labels, will cause the neural network to reward the TTPs with high support (such as T1124, T1571, and Mal-Sample) which still deliver relatively good performance even when training on a small percentage of the datasets. For instance, when sampling with just 5% percent of the training data:

1. T1124, T1571 and Mal-Sample are expected to have an average support of 3594.63, 4304.44, and 4654.65 in the training dataset.
2. All other TTPs (except T1135, which should still have a support $\simeq$ 18.11) should have a support value of less than 3.
3. The micro-average scores of all the TTPs that are not T1124, T1571 or Mal-Sample are equal to 0 in the vast majority of cases, even when given the TTP features as input.

Thus, the performance of STTP2STTP and S2STTP as the amount of training data reduces is likely to be dominated, not by the presence of TTP features, but by the presence of a few TTP labels with high support, especially at lower percentages. As a result, the performance of these two systems is likely to converge as the amount of training data is decreased, and have a starker difference as the amount of data is increased and the effect of the additional TTP features starts to dominate.

5.5.3 Adversarial Benign Flow Injection

In this last experiment, we consider the scenario in which an adversary tries to camouflage their malicious activity by masking it alongside various other benign activities in a given network traffic sample. Malicious actors often deploy such a tactic in order to trick any IDS into thinking that since the vast majority of the actions/behaviours performed by that malware are normal and routine, hence the sample must be benign. This in turn can enable the relatively few but malicious actions of said malware to execute unnoticed, and thus cause a security breach. We emulate such an attack on our system by randomly injecting varying amounts of benign traffic into known malware samples, and running them through our malware detection system without retraining them on any new data. This is meant to evaluate the robustness of our system to an unexpected but likely attack scenario, that is designed to evade our system's defences.

We test the robustness of both STTP2STTP and STTP2S in such a scenario, in the following manner:

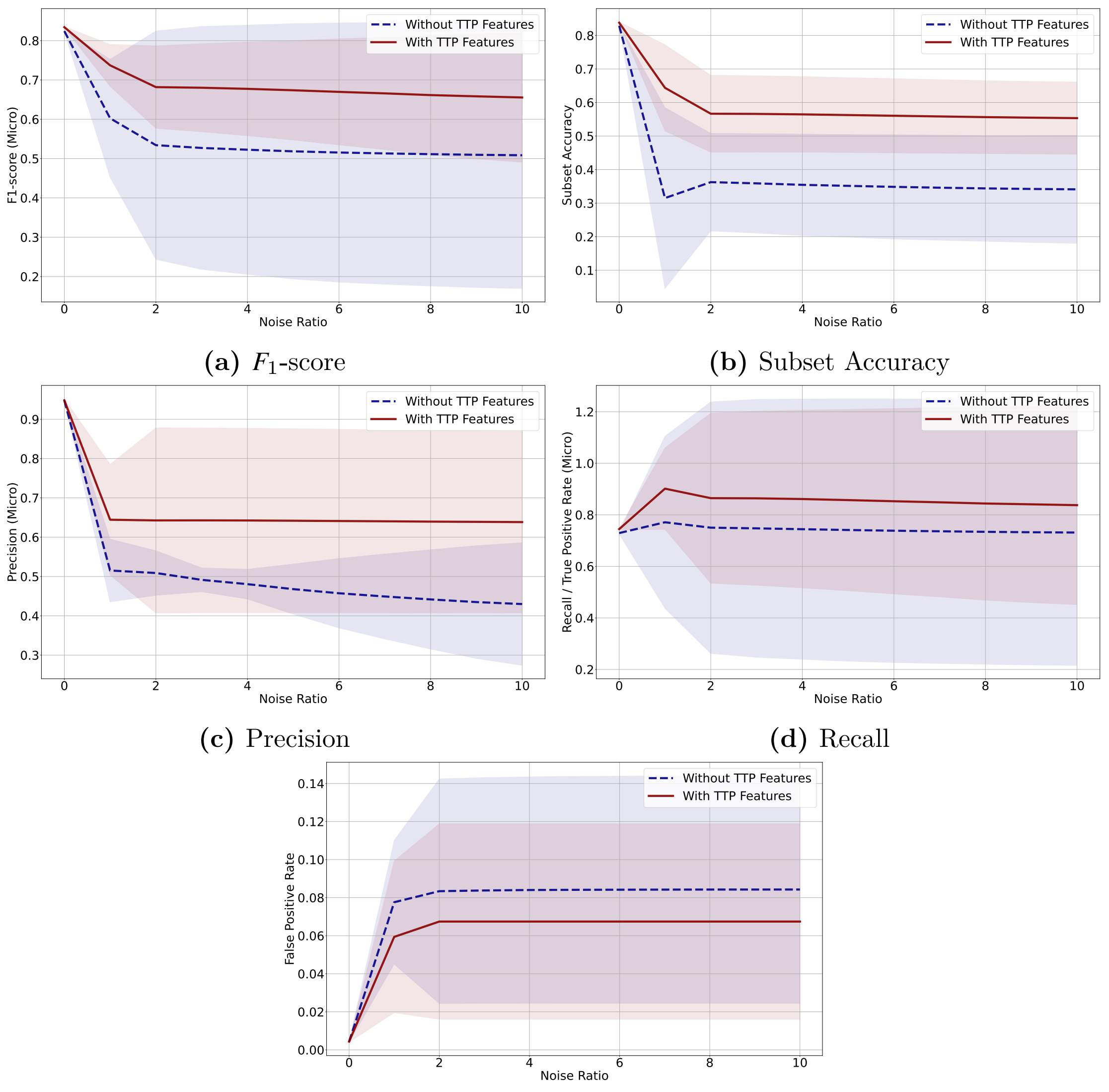
1. We take the benign traffic flows extracted from the Ember Benign dataset in Table 5.3, and use them as our source for injecting benign traffic into the malware samples.

2. To each malicious sample, we add a varying percentage of benign flows, starting from 0% all the way to 1000% at intervals of 100%. We thus create 10 new test sets, each corresponding to a particular percentage of camouflaged data injected into its malware samples.

3. We test the performance of STTP2STTP and S2STTP without retraining these systems on similar camouflaged data.

We measure the amount the adversary has camouflaged a particular malware sample, in terms of the Sample Noise Ratio (SNR), defined as the ratio of injected benign flows to malicious flows (thus, a SNR = 2 implies that for each malicious sample containing f flows, an additional $2f$ benign flows have been injected into it).

Given the setup of our experiment to simulate this adversarial attack, we expect that both STTP2STTP and S2STTP should have a decay in performance on the new camouflaged test sets, given that the newly generated samples have different characteristics compared to the samples used to train both systems. Additionally, we aim to inject noise up to a Sample Noise Ratio of 10 (implying there will be 10x more benign data than malicious data), thereby making the task of malware detection significantly harder. The questions we want to investigate here are: (*i*)



(a) F_1-score **(b)** Subset Accuracy

(c) Precision **(d)** Recall

(e) FPR

Figure 5.3: Performance of STTP2STTP and S2STTP when tested with varying percentages the noise ratio, defined as the ratio between the number of injected benign flows and the number of flows in the original sample.

whether TTP information is useful STTP2STTP in keeping the system's malware detection capabilities robust to attack, (*ii*) to what extent do STTP2STTP and S2STTP experience performance decay as a result of such an attack, and (*iii*) whether the performance gap between STTP2STTP and S2STTP increases or decreases with the Sample Noise Ratio.

The results of our experiment to simulate such an adversarial attack are shown in Figure 5.3. The figure shows the plots for the F_1-score, Subset Accuracy, Precision, Recall, and False Positive Rate, in their micro-average form on the Y-axis, while variation of the Sample Noise Ratio is shown on the X-axis.

We observe the following results from the figure:

1. STTP2STTP consistently outperforms S2STTP, implying that TTP features are indeed useful in improving the robustness of such a system to camouflage attacks.

2. The performance gap between STTP2STTP and S2STTP increases sharply up to a certain point, after which the difference becomes roughly constant or only marginally increases with an increase in the Sample Noise Ratio. This implies that using TTP features confers a consistent advantage to the robustness of the system, regardless of the amount of data injected.

3. Both systems experience a degradation in performance as the severity of the attack increases (i.e., an increase in the Sample Noise Ratio). However, this degradation in performance is significantly steep only up to SNR = 2, after which it plateaus. This suggests that there is a diminishing marginal utility when it comes to injecting benign flows, i.e., camouflage attacks past a 2:1 ratio of benign noise to actual malware traffic do not have a significantly better chance of success against our system.

4. Our results show a large standard deviation for both models, suggesting a high-degree of variability across the different runs of the experiment, which is to be expected when injecting random benign traffic into our test sets.

5.6 Discussion

Consistent Improvement over the State-of-the-Art. Our results highlight the consistency of our system's improvement over the state-of-the-art when it comes to detecting malware with the help of TTP features. Both versions of our system based on our multi-label DNN architecture (STTP2STTP and S2STTP) clearly outperform the binary DNN systems (STTP2S and S2S). This further highlights the power of our novel multi-label DNN architecture, since it enables us to outperform models like STTP2S that utilise TTP features, even when using a model like S2STTP which does not utilise TTP features.

Explainability and Precision in Detecting Malware. Our overall malware detection is not only better but also much more precise, as a result of our system being able to identify the malicious usage of each TTP individually. Instead of a binary output indicating the malicious or benign nature of a sample, our system can additionally identify which particular behaviours, as characterised by specific TTPs, were responsible for a malicious classification. Thus, making the output not only precise, but also more explainable to the user.

Recognition of rarely-occurring TTPs. ML-based malware detection systems rely on ample amounts of representative training data in order to reliably detect similar or perhaps unseen malware. Similarly, ML-based systems that rely on TTPs also require training on a significant number of datapoints for each TTP in order to later identify them. As a result, unlike other ML-based systems where TTPs with only a few datapoints might be overlooked during model training, our system can also accurately identify malicious behaviour using TTPs which only have a few datapoints present in the training data. As shown in our experiments, our system can identify malware using TTPs with as few as 15 datapoints with an average F_1-score of 0.946, as in the case of T1090.

System Configurability. A key part of our system design is that it can detect malicious usage of individual TTPs, thus allowing a user to choose the specific TTPs most relevant to their threat model. Additionally, we demonstrate that we can tune the detection thresholds for each TTP in our system, to enable even greater granularity of detection based on specific requirements. For instance, the threshold for a rarely-occurring but high-risk TTP can be lowered in a given environment, while a commonly-utilised TTP which does not pose a significant risk can have its detection threshold increased, thus increasing and reducing the number of positively detected samples respectively. Our results across our datasets highlight the benefits of this configurability—resulting in an average improvement of Macro F1-score by 12.5%, Macro Precision by 12%, and Macro Recall by 47.6%.

Utility of TTPs in Malware Detection. Our experimental analysis highlights the utility of TTP information in improving malware detection capabilities. When utilising our system with TTPs (sttp2sttp), we see an average F_1-score improvement of 1.08% at the sample level and 37.65% at the TTP-level, as compared to our system without TTPs (s2sttp). This can further be seen when comparing results at just the sample level (i.e., binary classification)—on average sttp2sttp outperforms s2sttp, sttp2s, and s2s across every metric (F_1-score, Accuracy, Precision, Recall, and FPR)—thus showing the utility of TTPs in improving standard malware detection capabilities.

Data Scarcity. We find that even when utilising limited training data, the addition of TTP features in our system architecture helps improve overall performance. Across the 30 experiments we performed (5 datasets x 6 training data percentages) sttp2sttp outperforms s2sttp 96.67% of the time as per their F_1-score. The addition of TTPs significantly improves the capability of the system even when using much less training data. For instance, sttp2sttp utilising 25% of the training data has a higher average F_1-score than s2sttp utilising 100% of the training data, and sttp2sttp utilising 10% of the training data has a higher average Recall than s2sttp utilising 100% of the training data.

Resilience to Adversarial Attack. Our experimental analysis also evaluates the capability of our system to resist adversarial attack, specifically camouflaging attacks, where an adversary deliberately injects benign flows into a malicious sample in order to camouflage its malicious activity. We measure this in terms of the Sample Noise Ratio (ratio of injected benign flows to malicious flows), and find that while system performance decreases as expected when the Noise Ratio increases, sttp2sttp is consistently able to perform better under these conditions than s2sttp. Indeed, from an average improvement of 1.21% in F_1-score when the Noise Ratio is 0.0, we see an average improvement of up to 28.93% in F_1-score when the Noise Ratio is 10.0. Similarly, we observe a consistent average improvement across every metric in the noisiest situation—62.17% in Subset Accuracy, 48.71% in Precision, 14.50% in Recall, and 20.23% in terms of FPR.

Minimal Additional Data. Our approach also highlights the minimal data requirements for achieving these significant performance gains. The TTPs we utilise in our system, as described in the MITRE ATT&CK framework, are open-source information, and our procedure to create TTP features using them does not require additional training data. Since these features can be created directly from existing data, utilising open-source domain knowledge regarding adversarial behaviour, it provides a vital performance benefit for the data-greedy machine learning algorithms.

Different Levels of Granularity. Our approach allows us to report the maliciousness of each sample at two levels of granularity which, as explained in [32], allows for different fine-grained outputs that can be used under divergent levels of expertise. In our case, we have just two levels (sample-level and TTP-level) but it is of course possible to (*i*) group the TTPs together according to the tactical goal of the attacker, (*ii*) subdivide each technique into the various sub-techniques it may include, and (*iii*) also include the procedures used to implement the technique or sub-technique. This would correspond to a five-level hierarchy reflecting the TTP ontology described in the MITRE ATT&CK framework, which, if incorporated into our model (thus having one hierarchical label per tactic, technique and procedure, plus the label for sample maliciousness) would give our system the ability to (*i*) report a more fine-grained output, and (*ii*) recognise a malicious activity at different levels of granularity, each with its own level of confidence. For instance, our system would be able to recognise a sample as malicious, e.g., at the tactical level while failing to recognise the specific technique and sub-technique or procedure being used: though not optimal, this would still be better than just reporting that the sample is malicious, or incorrectly detect the malicious usage of a specific technique and/or sub-technique and/or procedure.

5.7 Related Work

If we focus specifically on how TTPs have been used for the automatic detection of malicious activity, to the best of our knowledge, the only research literature that has explored this direction are our own recent works [162, 163] (as described in Chapters 3 and 4). In [162], the authors create an explainable and extensible IDS that exploits TTPs in order to detect malware. In this system, a different dataset is created for each TTP, where the label is considered positive if the sample is malicious and the TTP is present. For each dataset, a different decision tree [195] is trained and then used together with the others as an ensemble to decide on the maliciousness of the sample, thus resulting in an extensible and explainable system. However, the limitation of such an approach is that its modularity is achieved by having a separate ML engine for each TTP, thus sacrificing the possibility of exploiting the existing correlations between different TTPs to detect malicious activity (correlations among labels are often exploited to achieve better performance, see, e.g., [180, 153]). Additionally, this system conducts its analysis on a per-flow basis, which while effective, is unable to consider all the flows in a sample together and thus cannot evaluate the behaviour of a malware sample as a whole.

On the other hand, in [163] the authors create binary classification models, called TTPxML models, which incorporate TTP information as features into these models in order to detect network-based malware. They test this approach on a wide-variety of ML models, such as Support Vector Machines [46], Random Forests [80], and DNNs [117], and find that TTP features can be exploited to improve the malware detection capabilities of ML-based systems. Here, TTPs are first automatically extracted from the network sample, then a vectorial representation is created of both the sample features and the TTP features, which is subsequently provided as input to a binary classifier which labels the entire sample as malicious or not. This system is the state-of-the-art in terms of exploiting TTP information in an automated manner to detect network-based malware. Thus, we closely model this system in our sttp2s model, in order to be able to compare and contrast the results of our multi-label classification-based approach with the binary classification-based system described in that system.

5.8 Conclusions

In this chapter, we have designed a novel multi-label DNN architecture that detects malicious behaviour at two different levels of granularity—for the TTPs and the overall sample. By modelling this task as a multi-label classification task, we are able to detect malware more precisely on the basis of individual TTPs, improve on the state-of-the-art when it comes to detecting malicious behaviour when exploiting TTPs, and identify hard-to-detect malware which utilises uncommon or rarely-used TTPs. We back up our findings with an extensive comparative analysis, using a dataset of over 1.5 million real-world malware and benign samples. Lastly, we test the robustness of our system in various challenging conditions, such as when relying on limited training data or subjected to adversarial camouflage attacks, and demonstrate that it continues to outperform other comparable systems. Our experimental analysis demonstrates that such an architecture is indeed useful for improving the malware detection capabilities of a system (outperforming other comparable systems on both binary and multi-label malware classification tasks), and that it enables the effective exploitation of TTPs to further improve system performance (increasing the average F_1-score of the system by 1.08% at the sample level and 37.65% at the TTP-level when utilising TTPs in the detection pipeline).

The most important decision you make is to be in a good mood.

— Voltaire

6 Summary and Future Work

6.1 Summary

This thesis investigates methods of injecting human domain knowledge into purely data-driven systems meant for detecting malware. We do this by exploiting TTPs from the MITRE ATT&CK framework, and making the knowledge of adversarial behaviour they correspond to, accessible to malware detection systems.

We first develop a specialised ML-based NIDS called RADAR that is designed to exploit TTPs in an automated manner to detect malicious activity in network traffic, and hence classify malware. RADAR is the first malware detection system with these capabilities. It is designed keeping in mind the principles of extensibility and explainability, which are critically important in the context of malware analysis. We evaluate our system on a large dataset comprising over 2.2 million real-world malware and benign samples, and find that it indeed effective in detecting a broad range of malware. Our results also highlight the fact that TTPs can be correspond to behaviour in both malicious and benign samples, thereby necessitating the need for specifically detecting the malicious usage of these TTPs. These conclusions form the fundamental basis of our research going forward in this area of malware detection.

Next, we look at whether such malware detection capabilities can be reproduced for any generic ML-based NIDS rather than just a specialised one, whose malware detection pipeline is designed to incorporate TTPs. To this end, we create a methodology for transforming TTP information into TTP features, which can be exploited by various machine learning algorithms. Unlike RADAR, this method classifies malware at the sample-level (rather than the flow-level) by creating an aggregated representation of these NetFlows, known as a Bag of Flows. These models that can automatically exploit TTP information as features, called TTPxML models, are then comparatively evaluated with their standard counterparts. We test them over a large balanced dataset of over 1.5 million malicious and benign samples, and find that the TTPxML models outperform the standard models in nearly all cases and for every type of ML model. This for the first time presents the possibility that TTP information can be exploited by any existing ML-based NIDS, thus allowing for an upgrade in the malware detection capabilities of any organisation's security architecture.

Finally, we go one step further and attempt to increase the granularity and precision of our TTP-based malware detection capabilities. We design a methodology that models the task of network-based malware detection as a multi-label classification problem that allows us to detect malware on the basis of individual TTPs, as well as collectively at the sample level. Our methodology further generalises the approach we propose in the previous chapter, and presents a framework for comparing these various approaches. We rigorously evaluate these various approaches over a large balanced dataset comprising over 1.5 million malicious and benign samples, and find that our multi-label model that exploits TTP information is able to outperform all other approaches. We find that beside improved performance, there are definitive benefits to using such a multi-label approach, such as: (*i*) being able to detect malware more precisely on the basis of individual TTPs, (*ii*) being able to better detect malicious behaviour corresponding to uncommon or rarely-utilised TTPs, and (*iii*) being able to tweak the detection thresholds for individual TTPs, thus allowing greater flexibility in the types of malicious activity that the system can detect.

Overall, this thesis presents a step forward in the area of network-based malware detection by exploiting existing human domain knowledge. We believe such an approach will be increasingly important in the future, as the prevalence and capabilities of automated systems keeps increasing, so should their ability to utilise the knowledge of human expertise in the field. With the exponential rise of ML and AI based systems, we now rely a lot more on machine expertise for various tasks. Perhaps it is time to realise the opposite is equally important—human expertise in many domains is vastly superior, and we can leverage the power of automated learning-based systems far more effectively if we lend them our knowledge.

6.2 Datasets

To further facilitate research in this domain, we have publicly released the datasets we have utilised in this thesis. The repository with our datasets can be found here: https://github.com/baddymaster/BOF24. The creation methodology, data sources used, dataset statistics, and the desiderata for the dataset are described in Chapters 4 and 5.

This should hopefully provide valuable data to security researchers, since it contains 5 balanced datasets containing over 1.5 million malicious and benign samples from a diverse set of sources. In addition, our dataset contains both network traffic features and TTP features in the Bag of Flow representation, thus providing more data that can be used to benchmark future systems or detect novel strains of malware.

6.3 Future Work

Based on the findings of our research, we have identified a number of directions for future research in the domain of malware analysis that exploits domain knowledge to improve a system's capabilities. Broadly, these can be divided into two categories: (*i*) extending malware analysis capabilities within the context of network-based malware detection, and (*ii*) expanding this novel form of malware analysis and the methodologies proposed here to the domain of host-based malware detection.

Within the context of network-based malware analysis, there are several avenues for extending our research. For a specialised NIDS like RADAR, it would be interesting to experiment with other malware detection policies that are tuned for a more specific use-case, e.g., by assigning weights to the TTPs and tuning them on the basis of the behaviours we want to detect. In the case of the multi-label DNN architecture based approach, it would be useful to expand the hierarchical representation of TTPs used to include other levels of reasoning, such as at the level of tactics and procedures. This would enable a much richer hierarchical representation, thus allowing for a finer level of granularity when it comes to both detecting and reasoning about malware. More generally, it would be quite interesting to increase the number of TTPs that can detected by these systems. Adding additional TTPs should: (*i*) further increase the coverage of malicious activities that our systems are able to reason about, (*ii*) correspond to a better and more accurate profiling of each sample, and thus (*iii*) further increase system performance.

With regard to the latter, there would be tremendous benefit in adapting our approach and system design for host-based malware. There are several reasons why supporting host-based malware would be useful: (*i*) the vast majority of TTPs in the ATT&CK framework are designed to be detected, or are easier to detect, using host-based data, (*ii*) host-based data is richer and has more granularity, making it ideal for data-driven approaches like the ones we utilise, and (*iii*) the accuracy with which TTPs can be detected using host-based data is far higher, thus reducing false positives. Although implementing malware detection at the host-level presents certain challenges, the advantages of applying our methods in this area, which would capture a wider array of malicious activities, would be tremendously valuable for those defending against malware attacks. Additionally, a host-based approach would naturally complement the capabilities of our network-based intrusion detection systems, and provide greater protection against a wider variety of threats.

At a broader level, there are plenty of opportunities to extend our research further. For one, the ontology of human domain knowledge we consider in our work is specifically the MITRE ATT&CK framework. Future research could expand or substitute this knowledge base with other ontologies such as Lockheed Martin's Cyber Kill Chain [116] or the Diamond Model [28]. These would presumably offer their own set of trade-offs when compared to the ATT&CK framework, and as such might be advantageous to employ in certain contexts. Secondly, the task we have utilised TTP information for is malware detection. However, further extensions of our research could look at using TTP information to perform and improve other related malware analysis tasks, such as malware type classification and malware family classification.

Lastly, beyond a purely research perspective there is the opportunity to deploy one or more of our systems and methodologies to a real-world network and expand their applicability to such an environment. Our work is well-suited for such a task given that is was designed keeping in mind properties well-suited for an enterprise environment: (i) lightweight and efficient, (ii) privacy-preserving, and (iii) able to analyse arbitrary network traffic independent of traffic type, protocol, or network topology. Our system design choices ensure that our approach can easily adapt to operate effectively in a new environment. To make sure this happens, the ML/DL models used in our systems would require training on sufficient and representative network traffic data for a given enterprise. The policies and thresholds that our systems use could further be configured and tuned according to the particulars of the given network environment and their threat model. A further benefit of our approach in this context would be that the TTP information that we have curated would be immediately usable and applicable to a brand-new environment. Thus, there is significant scope for improving malware detection capabilities and performance, if our work combining human domain knowledge with data-driven malware detection approaches is deployed in real-world environments.

In conclusion, given the unique capabilities, high configurability, and adaptability of the systems proposed in this thesis, it is easy to see how they could benefit not only future academic researchers but also cyber security practitioners defending against real-world threats.

Plagiarism: Getting into trouble for something you didn't do.

— Anonymous

Bibliography

[1] DARPA Intrusion Detection Evaluation Dataset 1999 | MIT Lincoln Laboratory. URL `https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset.html:html` .

[2] Static Analysis with CodeSonar | GrammaTech. URL `https://www.grammatech.com/products/source-code-analysis`.

[3] Ghidra. URL `https://www.nsa.gov/resources/everyone/ghidra/`.

[4] IDA Pro – Hex Rays. URL `https://www.hex-rays.com/products/ida/`.

[5] KDD Cup 1999 Data. URL `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

[6] LordPE 1.31. URL `https://www.softpedia.com/get/Programming/File-Editors/LordPE.shtml` .

[7] OllyDumpEx. URL `https://low-priority.appspot.com/ollydumpex/`.

[8] Zeeshan Akram, Mamoona Majid, and Shaista Habib. A Systematic Literature Review: Usage of Logistic Regression for Malware Detection. In *2021 International Conference on Innovative Computing (ICIC)*, pages 1–8, 2021.

[9] Rawan Al-Shaer, Jonathan M Spring, and Eliana Christou. Learning the Associations of MITRE ATT&CK Adversarial Techniques. In *IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020.

[10] Bushra A AlAhmadi and Ivan Martinovic. Malclassifier: Malware Family Classification Using Network Flow Sequence Behaviour. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–13. IEEE, 2018.

[11] Bushra Abdulrahman AlAhmadi. *Malware Detection in Security Operation Centres*. PhD thesis, University of Oxford, 2019.

[12] Nahla Ben Amor, Salem Benferhat, and Zied Elouedi. Naive Bayes vs Decision Trees in Intrusion Detection Systems. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 420–424, 2004.

[13] Hyrum S. Anderson and Phil Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *CoRR*, abs/1804.04637, 2018.

[14] Ayesha Binte Ashfaq, Zainab Abaid, Maliha Ismail, Muhammad Umar Aslam, Affan A Syed, and Syed Ali Khayam. Diagnosing Bot Infections Using Bayesian Inference. *Journal of Computer Virology and Hacking Techniques*, 14(1):21–38, 2018.

[15] Ömer Aslan and Refik Samet. A Comprehensive Review on Malware Detection Approaches. *IEEE Access*, 8:6249–6271, 2020.

[16] Martin Atzmueller and Rushed Kanawati. Explainability in Cyber Security using Complex Network Analysis: A Brief Methodological Overview. In David Megías, Roberto Di Pietro, and Joaquín García-Alfaro, editors, *European Interdisciplinary Cybersecurity Conference*, pages 49–52. ACM, 2022.

[17] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., USA, 1999. ISBN 020139829X.

[18] Michael Bailey, Jon Oberheide, Jon Andersen, Z Morley Mao, Farnam Jahanian, and Jose Nazario. Automated Classification and Analysis of Internet Malware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 178–197. Springer, 2007.

[19] Rohan Bapat, Abhijith Mandya, Xinyang Liu, Brendan Abraham, Donald Brown, Hyojung Kang, and Malathi Veeraraghavan. Identifying Malicious Botnet Traffic Using Logistic Regression. In *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pages 266–271, 2018. doi: 10.1109/SIEDS.2018.8374749.

[20] Karel Bartos, Michal Sofka, and Vojtech Franc. Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 807–822. USENIX Association, 2016.

[21] G Bassett, D Hylender, P Langlois, A Pinto, and S Widup. Data Breach Investigations Report, 2021.

[22] Ulrich Bayer. *TTAnalyze: A Tool for Analyzing Malware*. PhD thesis, 2005.

[23] Ulrich Bayer, Imam Habibi, Davide Balzarotti, and Engin Kirda. A View on Current Malware Behaviors. In Wenke Lee, editor, *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '09, Boston, MA, USA, April 21, 2009*. USENIX Association, 2009.

[24] Dmitri Bekerman, Bracha Shapira, Lior Rokach, and Ariel Bar. Unknown malware detection using network traffic classification. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015*, pages 134–142. IEEE, 2015.

[25] Sweta Bhattacharya, Siva Rama Krishnan S, Praveen Kumar Reddy Maddikunta, Rajesh Kaluri, Saurabh Singh, Thippa Reddy Gadekallu, Mamoun Alazab, and Usman Tariq. A Novel PCA-Firefly Based XGBoost Classification Model for Intrusion Detection in Networks Using GPU. *Electronics*, 9(2), 2020. ISSN 2079-9292.

[26] Leyla Bilge, Davide Balzarotti, William K. Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: Detecting Botnet Command and Control Servers through Large-scale NetFlow Analysis. In Robert H'obbes' Zakon, editor, *28th Annual Computer Security Applications Conference, ACSAC 2012, Orlando, FL, USA, 3-7 December 2012*, pages 129–138. ACM, 2012.

[27] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. Detecting Self-mutating Malware Using Control-Flow Graph Matching. In Roland Büschkes and Pavel Laskov, editors, *Detection of Intrusions and Malware & Vulnerability Assessment, Third International Conference, DIMVA 2006, Berlin, Germany, July 13-14, 2006, Proceedings*, volume 4064 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2006.

[28] Sergio Caltagirone, Andrew Pendergast, and Christopher Betz. The Diamond Model of Intrusion Analysis. *Threat Connect*, 298(0704):1–61, 2013.

[29] James Cannady. Artificial Neural Networks for Misuse Detection. In *National information systems security conference*, volume 26, pages 443–456. Baltimore, 1998.

[30] Nicola Capuano, Giuseppe Fenza, Vincenzo Loia, and Claudio Stanzione. Explainable Artificial Intelligence in CyberSecurity: A Survey. *IEEE Access*, 10:93575–93600, 2022.

[31] Lorenzo Cavallaro, Christopher Kruegel, and Giovanni Vigna. Mining the Network Behavior of Bots. *Technical Report 2009-12*, 2009.

[32] Dongliang Chang, Kaiyue Pang, Yixiao Zheng, Zhanyu Ma, Yi-Zhe Song, and Jun Guo. Your "Flamingo" is My "Bird": Fine-Grained, or Not. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 11476–11485. Computer Vision Foundation IEEE, 2021.

[33] Duen Horng "Polo" Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 131–142. SIAM, 2011.

[34] Hao Chen and David A. Wagner. MOPS: an infrastructure for examining security properties of software. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 235–244. ACM, 2002.

[35] Hao Chen, Drew Dean, and David A. Wagner. Model Checking One Million Lines of C Code. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2004, San Diego, California, USA*. The Internet Society, 2004.

[36] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794. ACM, 2016.

[37] Xu Chen, Jonathon Andersen, Zhuoqing Morley Mao, Michael D. Bailey, and Jose Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *The 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2008, June 24-27, 2008, Anchorage, Alaska, USA, Proceedings*, pages 177–186. IEEE Computer Society, 2008.

[38] Veronica Chierzi and Fernando Mercês. Evolution of IoT Linux Malware: A MITRE ATT&CK TTP Based Approach. In *2021 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–11, 2021.

[39] Mihai Christodorescu and Somesh Jha. Static Analysis of Executables to Detect Malicious Patterns. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003.

[40] Mihai Christodorescu and Somesh Jha. Testing Malware Detectors. *ACM SIGSOFT Software Engineering Notes*, 29(4):34–44, jul 2004. ISSN 0163-5948.

[41] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Xiaodong Song, and Randal E. Bryant. Semantics-Aware Malware Detection. In *2005 IEEE Symposium on Security and Privacy (S&P 2005), 8-11 May 2005, Oakland, CA, USA*, pages 32–46. IEEE Computer Society, 2005.

[42] Mihai Christodorescu, Somesh Jha, and Christopher 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*, pages 5–14, 2007.

[43] Benoit Claise, Brian Trammell, and Paul Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. *RFC*, 7011:1–76, 2013.

[44] Pierre Comon. Independent Component Analysis, A New Concept? *Signal Process.*, 36(3):287–314, 1994.

[45] Carlos Garcia Cordero, Sascha Hauke, Max Mühlhäuser, and Mathias Fischer. Analyzing flow-based anomaly intrusion detection using Replicator Neural Networks. In *14th Annual Conference on Privacy, Security and Trust, PST 2016, Auckland, New Zealand, December 12-14, 2016*, pages 317–324. IEEE, 2016.

[46] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 1995.

[47] David R Cox. The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society*, 1958.

[48] Roman Daszczyszak, Dan Ellis, Steve Luke, and Sean Whitley. TTP-Based Hunting. Technical report, MITRE Corp McLean VA, 2019.

[49] Janez Demsar. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[50] Dorothy E Denning. An Intrusion-Detection Model. *IEEE Transactions on software engineering*, (2):222–232, 1987.

[51] Sukhpreet Singh Dhaliwal, Abdullah Al Nahid, and Robert Abbas. Effective Intrusion Detection System Using XGBoost. *Information*, 9(7):149, 2018.

[52] Dinil Mon Divakaran, Le Su, Yung Siang Liau, and Vrizlynn L. L. Thing. SLIC: Self-Learning Intelligent Classifier for network traffic. *Computer Networks*, 91:283–297, 2015.

[53] Emily H. Do and Vijay Gadepally. Classifying Anomalies for Network Security. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 2907–2911. IEEE, 2020.

[54] Salijona Dyrmishi, Salah Ghamizi, Thibault Simonetto, Yves Le Traon, and Maxime Cordy. On The Empirical Effectiveness of Unrealistic Adversarial Hardening Against Realistic Adversarial Attacks. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 1384–1400. IEEE, 2023.

[55] Manuel Egele, Christopher Kruegel, Engin Kirda, Heng Yin, and Dawn Xiaodong Song. Dynamic Spyware Analysis. In Jeff Chase and Srinivasan Seshan, editors, *Proceedings of the 2007 USENIX Annual Technical Conference, Santa Clara, CA, USA, June 17-22, 2007*, pages 233–246. USENIX, 2007.

[56] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A Survey on Automated Dynamic Malware-Analysis Techniques and Tools. *ACM Computing Surveys*, 44(2):6:1–6:42, 2012.

[57] Jeffrey Fairbanks, Andres Orbe, Christine Patterson, Janet Layne, Edoardo Serra, and Marion Scheepers. Identifying ATT&CK Tactics in Android Malware Control Flow Graph Through Graph Representation Learning and Interpretability. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5602–5608, 2021.

[58] Dewan Md Farid, Nouria Harbi, and Mohammad Zahidur Rahman. Combining Naive Bayes and Decision Tree for Adaptive Intrusion Detection. *arXiv preprint arXiv:1005.4496*, 2010.

[59] Nabila Farnaaz and MA Jabbar. Random Forest Modeling for Network Intrusion Detection System. *Procedia Computer Science*, 89(1):213–217, 2016.

[60] Henry Hanping Feng, Jonathon T. Giffin, Yong Huang, Somesh Jha, Wenke Lee, and Barton P. Miller. Formalizing Sensitivity in Static Analysis for Intrusion Detection. In *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, page 194. IEEE Computer Society, 2004.

[61] Jiayin Feng, Limin Shen, Zhen Chen, Yuying Wang, and Hui Li. A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic. *IEEE Access*, 8:125786–125796, 2020.

[62] Lei Feng, Bo An, and Shuo He. Collaboration Based Multi-Label Learning. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3550–3557. AAAI Press, 2019.

[63] Majid Ghonji Feshki, Omid Sojoodi Shijani, and Minoo Deljavan Anvary. Managing Intrusion Detection Alerts Using Support Vector Machines. *International Journal of Computer Science and Security (IJCSS)*, 9(5):266, 2015.

[64] Prahlad Fogla, Monirul Islam Sharif, Roberto Perdisci, Oleg M. Kolesnikov, and Wenke Lee. Polymorphic Blending Attacks. In Angelos D. Keromytis, editor, *Proceedings of the 15th USENIX Security Symposium, Vancouver, BC, Canada, July 31 - August 4, 2006*. USENIX Association, 2006.

[65] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*, 1995.

[66] Savan Gadhiya, Kaushal Bhavsar, and PD Student. Techniques for malware analysis. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(4):2277–128, 2013.

[67] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. Malware Analysis and Classification: A Survey. *Journal of Information Security*, 2014, 2014.

[68] Pedro Garcia-Teodoro, Jesús Esteban Díaz Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges. *Computers & Security*, 28(1-2):18–28, 2009.

[69] Daniel Gibert, Carles Mateu, and Jordi Planes. The Rise of Machine Learning for Detection and Classification of Malware: Research Developments, Trends and Challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[70] Eleonora Giunchiglia and Thomas Lukasiewicz. Coherent hierarchical multi-label classification networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[71] Eleonora Giunchiglia and Thomas Lukasiewicz. Multi-Label Classification Neural Networks with Hard Logical Constraints. *Journal of Artificial Intelligence Research*, 72:759–818, 2021.

[72] Eleonora Giunchiglia, Fergus Imrie, Mihaela van der Schaar, and Thomas Lukasiewicz. Machine Learning with Requirements: a Manifesto. *CoRR*, abs/2304.03674, 2023.

[73] Christian Gorecki, Felix C. Freiling, Marc Kührer, and Thorsten Holz. TrumanBox: Improving Dynamic Malware Analysis by Emulating the Internet. *Lecture Notes in Computer Science*, 6976 LNCS:208–222, 2011.

[74] Martin Grill, Ivan Nikolaev, Veronica Valeros, and Martin Rehák. Detecting DGA Malware Using NetFlow. In Remi Badonnel, Jin Xiao, Shingo Ata, Filip De Turck, Voicu Groza, and Carlos Raniery Paula dos Santos, editors, *IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, Ottawa, ON, Canada, 11-15 May, 2015*, pages 1304–1309. IEEE, 2015.

[75] Guofei Gu, Phillip A. Porras, Vinod Yegneswaran, Martin W. Fong, and Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In Niels Provos, editor, *Proceedings of the 16th USENIX Security Symposium, Boston, MA, USA, August 6-10, 2007*. USENIX Association, 2007.

[76] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 139–154. USENIX Association, 2008.

[77] 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, San Diego, California, USA, 10th February - 13th February 2008*. The Internet Society, 2008.

[78] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang Yang. XAI - Explainable Artificial Intelligence. *Science Robotics*, 2019.

[79] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical Provenance Analysis for Endpoint Detection and Response Systems. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1172–1189. IEEE, 2020.

[80] Tin Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Pattern Analysis Machine Intelligence*, 20(8):832–844, 1998.

[81] Mohammad Sazzadul Hoque, Md Mukit, Md Bikas, Abu Naser, et al. An Implementation of Intrusion Detection System Using Genetic Algorithm. *arXiv preprint arXiv:1204.1336*, 2012.

[82] Shi-Jinn Horng, Ming-Yang Su, Yuan-Hsin Chen, Tzong-Wann Kao, Rong-Jian Chen, Jui-Lin Lai, and Citra Dwi Perkasa. A Novel Intrusion Detection System Based on Hierarchical Clustering and Support Vector Machines. *Expert systems with Applications*, 38(1):306–313, 2011.

[83] Huajun Huang, Liang Qian, and Yaojun Wang. A SVM-based Technique to Detect Phishing URLs. *Information Technology Journal*, 11:921–925, 2012.

[84] Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. TTPDrill: Automatic and Accurate Extraction of Threat Actions from Unstructured Text of CTI Sources. In *Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017*, pages 103–115. ACM, 2017.

[85] Koral Ilgun, Richard A Kemmerer, and Phillip A Porras. State Transition Analysis: A Rule-based Intrusion Detection Approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, 1995.

[86] Christopher Inacio and Brian Trammell. YAF: yet another flowmeter. In Rudi van Drunen, editor, *Uncovering the Secrets of System Administration: Proceedings of the 24th Large Installation System Administration Conference, LISA 2010, San Jose, CA, USA, November 7-12, 2010*. USENIX Association, 2010.

[87] Van Jacobson. Tcpdump. *ftp://ftp.ee.lbl.gov*, 1989. URL `https://www.tcpdump.org/`.

[88] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson Prentice Hall, 2008.

[89] Mina Esmail Zadeh Nojoo Kambar, Armin Esmaeilzadeh, Yoohwan Kim, and Kazem Taghva. A Survey on Mobile Malware Detection Methods using Machine Learning. In *12th IEEE Annual Computing and Communication Workshop and Conference, CCWC 2022, Las Vegas, NV, USA, January 26-29, 2022*, pages 215–221. IEEE, 2022.

[90] Min Gyung Kang, Pongsin Poosankam, and Heng Yin. Renovo: A Hidden Code Extractor for Packed Executables. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode*, WORM '07, page 46–53, New York, NY, USA, 2007. Association for Computing Machinery.

[91] Chaouki Khammassi and Saoussen Krichen. A GA-LR Wrapper Approach for Feature Selection in Network Intrusion Detection. *Computers & Security*, 70:255–277, 2017.

[92] Firoz Khan, Jinesh Ahamed, Seifedine Kadry, and Lakshmana Ramasamy. Detecting Malicious URLs Using Binary Classification through AdaBoost Algorithm. *International Journal of Electrical and Computer Engineering*, 10: 997–1005, 2020.

[93] Hyang-Ah Kim and Brad Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In Matt Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 271–286. USENIX, 2004.

[94] Johannes Kinder, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. Detecting Malicious Code by Model Checking. In Klaus Julisch and Christopher Krügel, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment, Second International Conference, DIMVA 2005, Vienna, Austria, July 7-8, 2005, Proceedings*, volume 3548 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 2005.

[95] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[96] Engin Kirda and Christopher Kruegel. Behavior-based Spyware Detection. In Angelos D. Keromytis, editor, *Proceedings of the 15th USENIX Security Symposium, Vancouver, BC, Canada, July 31 - August 4, 2006*. USENIX Association, 2006.

[97] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang. Effective and Efficient Malware Detection at the End Host. In Fabian Monrose, editor, *18th USENIX Security Symposium, Montreal, Canada, August 10-14, 2009, Proceedings*, pages 351–366. USENIX Association, 2009.

[98] J Zico Kolter and Marcus A Maloof. Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research*, 7(Dec): 2721–2744, 2006.

[99] Christopher Kruegel and Thomas Toth. Using Decision Trees to Improve Signature-based Intrusion Detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 173–191. Springer, 2003.

[100] Christopher Krügel, William K. Robertson, and Giovanni Vigna. Detecting Kernel-Level Rootkits Through Binary Analysis. In *20th Annual Computer Security Applications Conference (ACSAC 2004), 6-10 December 2004, Tucson, AZ, USA*, pages 91–100. IEEE Computer Society, 2004.

[101] Christopher Krügel, Engin Kirda, Darren Mutz, William K. Robertson, and Giovanni Vigna. Polymorphic Worm Detection Using Structural Information of Executables. In Alfonso Valdes and Diego Zamboni, editors, *Recent Advances in Intrusion Detection, 8th International Symposium, RAID 2005, Seattle, WA, USA, September 7-9, 2005, Revised Papers*, volume 3858 of *Lecture Notes in Computer Science*, pages 207–226. Springer, 2005.

[102] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[103] Sushil Kumar et al. An Emerging Threat Fileless Malware: A Survey and Research Challenges. *Cybersecurity*, 3(1):1–12, 2020.

[104] Aditya Kuppa, Slawomir Grzonkowski, Muhammad Rizwan Asghar, and Nhien-An Le-Khac. Finding Rats in Cats: Detecting Stealthy Attacks using Group Anomaly Detection. In *18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 13th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2019, Rotorua, New Zealand, August 5-8, 2019*, pages 442–449. IEEE, 2019.

[105] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C. Suh, Ikkyun Kim, and Kuinam J. Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, 22(Suppl 1):949–961, 2019.

[106] Wenke Lee and Salvatore J. Stolfo. Data Mining Approaches for Intrusion Detection. In Aviel D. Rubin, editor, *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29, 1998*. USENIX Association, 1998.

[107] Valentine Legoy, Marco Caselli, Christin Seifert, and Andreas Peter. Automated Retrieval of ATT&CK Tactics and Techniques for Cyber Threat Reports. *preprint arXiv:2004.14322*, 2020.

[108] Wei-Jen Li, Ke Wang, Salvatore J Stolfo, and Benjamin Herzog. Fileprints: Identifying File Types by n-gram Analysis. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, pages 64–71. IEEE, 2005.

[109] Wei-Jen Li, Salvatore J. Stolfo, Angelos Stavrou, Elli Androulaki, and Angelos D. Keromytis. A Study of Malcode-Bearing Documents. In Bernhard M. Hämmerli and Robin Sommer, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment, 4th International Conference, DIMVA 2007, Lucerne, Switzerland, July 12-13, 2007, Proceedings*, volume 4579 of *Lecture Notes in Computer Science*, pages 231–250. Springer, 2007.

[110] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise. In Lorenzo Cavallaro, Johannes

Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1777–1794. ACM, 2019.

[111] Raymond W Lo, Karl N Levitt, and Ronald A Olsson. MCF: A Malicious Code Filter. *Computers & Security*, 14(6):541–566, 1995.

[112] Ritesh Malaiya, Donghwoon Kwon, Sang Suh, Hyunjoo Kim, Ikkyun Kim, and Jinoh Kim. An Empirical Evaluation of Deep Learning for Network Anomaly Detection. *IEEE Access*, 7:140806–140817, 2019.

[113] Malwarebytes. Malwarebytes 2022 Threat Review, 2022. URL `https://www.malwarebytes.com/resources/malwarebytes-threat-review-2022/index.html`.

[114] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 2001.

[115] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. OmniUnpack: Fast, Generic, and Safe Unpacking of Malware. In *23rd Annual Computer Security Applications Conference (ACSAC 2007), December 10-14, 2007, Miami Beach, Florida, USA*, pages 431–441. IEEE Computer Society, 2007.

[116] Lockheed Martin. Cyber Kill Chain | Lockheed Martin. URL `https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html`.

[117] Warren McCulloch and Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The bulletin of mathematical biophysics*, 1943.

[118] Gary McGraw and Greg Morrisett. Attacking Malicious Code: A Report to the Infosec Research Council. *IEEE Software*, 17(5):33–41, 2000.

[119] Michael MchPhee. Methods to Employ Zeek in Detecting MITRE ATT&CK Techniques. Technical report, SANS Institute, July 2020.

[120] Hesham Mekky, Aziz Mohaisen, and Zhi-Li Zhang. Separation of benign and malicious network events for accurate malware family classification. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015*, pages 125–133. IEEE, 2015.

[121] Otgonpurev Mendsaikhan, Hirokazu Hasegawa, Yukiko Yamaguchi, and Hajime Shimada. Automatic Mapping of Vulnerability Information to Adversary Techniques. In *The 14th International Conference on Emerging Security Information, Systems and Tech.*, 2020.

[122] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and Venkat N. Venkatakrishnan. ProPatrol: Attack Investigation via Extracted High-Level Tasks. In Vinod Ganapathy, Trent Jaeger, and R. K. Shyamasundar, editors, *Information Systems Security - 14th International Conference, ICISS 2018, Bangalore, India, December 17-19, 2018, Proceedings*, volume 11281 of *Lecture Notes in Computer Science*, pages 107–126. Springer, 2018.

[123] Sadegh Momeni Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V. N. Venkatakrishnan. HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1137–1152. IEEE, 2019.

[124] David L. Mills. Network time protocol (NTP). Technical report, 1985.

[125] MITRE Corporation. BZAR: A Set of Zeek Scripts to Detect ATT&CK Techniques., Sep 2020. URL `https://github.com/mitre-attack/bzar`.

[126] MITRE Corporation. MITRE ATT&CK, 2022. URL `https://attack.mitre.org/`.

[127] Aziz Mohaisen, Andrew G. West, Allison Mankin, and Omar Alrawi. Chatter: Classifying Malware Families using System Event Ordering. In *IEEE Conference on Communications and Network Security, CNS 2014, San Francisco, CA, USA, October 29-31, 2014*, pages 283–291. IEEE, 2014. doi: 10.1109/CNS.2014.6997496. URL `https://doi.org/10.1109/CNS.2014.6997496`.

[128] Andreas Moser, Christopher Krüegel, and Engin Kirda. Limits of Static Analysis for Malware Detection. In *23rd Annual Computer Security Applications Conference (ACSAC 2007), December 10-14, 2007, Miami Beach, Florida, USA*, pages 421–430. IEEE Computer Society, 2007.

[129] Andreas Moser, Christopher Krügel, and Engin Kirda. Exploring Multiple Execution Paths for Malware Analysis. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pages 231–245. IEEE Computer Society, 2007.

[130] Srijith K Nair, Patrick ND Simpson, Bruno Crispo, and Andrew S Tanenbaum. A Virtual Machine Based Information Flow Control System for Policy Enforcement. *Electronic Notes in Theoretical Computer Science*, 197(1): 3–16, 2008.

[131] Pejman Najafi, Alexander Mühle, Wenzel Pünter, Feng Cheng, and Christoph Meinel. MalRank: A Measure of Maliciousness in SIEM-based Knowledge Graphs. In David Balenson, editor, *Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC 2019, San Juan, PR, USA, December 09-13, 2019*, pages 417–429. ACM, 2019.

[132] Saeed Nari and Ali A. Ghorbani. Automated Malware Classification Based on Network Behavior. In *International Conference on Computing, Networking and Communications, ICNC 2013, San Diego, CA, USA, January 28-31, 2013*, pages 642–647. IEEE Computer Society, 2013.

[133] Neo4J. Neo4J Graph Database, 2022. URL `https://neo4j.com/product/neo4j-graph-database/`.

[134] James Newsome and Dawn Xiaodong Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005, San Diego, California, USA*. The Internet Society, 2005.

[135] James Newsome, Brad Karp, and Dawn Xiaodong Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *2005 IEEE Symposium on Security and Privacy (S&P 2005), 8-11 May 2005, Oakland, CA, USA*, pages 226–241. IEEE Computer Society, 2005.

[136] Editors of Encyclopaedia Britannica. Hydra. *Encyclopedia Britannica*, 2023. URL `https://www.britannica.com/topic/Hydra-Greek-mythology`.

[137] Kris Oosthoek and Christian Doerr. Sok: Att&ck techniques and trends in windows malware. In Songqing Chen, Kim-Kwang Raymond Choo, Xinwen Fu, Wenjing Lou, and Aziz Mohaisen, editors, *Security and Privacy in Communication Networks - 15th EAI International Conference, SecureComm 2019, Orlando, FL, USA, October 23-25, 2019, Proceedings, Part I*, volume 304 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 406–425. Springer, 2019.

[138] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Elsevier, 2006.

[139] Mrutyunjaya Panda and Manas Ranjan Patra. Network Intrusion Detection Using Naive Bayes. *International Journal of Computer Science and Network Security*, 7(12):258–263, 2007.

[140] Animesh Patcha and Jung-Min Park. An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends. *Computer networks*, 51(12):3448–3470, 2007.

[141] David Patten. The evolution to fileless malware. *Retrieved from `https://www.coursehero.com/file/93814953/DPatten-Filelesspdf/`*, 2017.

[142] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In Aviel D. Rubin, editor, *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29, 1998*. USENIX Association, 1998.

[143] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[144] Chengwei Peng, Xiaochun Yun, Yongzheng Zhang, Shuhao Li, and Jun Xiao. Discovering Malicious Domains through Alias-Canonical Graph. In *2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, August 1-4, 2017*, pages 225–232. IEEE Computer Society, 2017.

[145] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*, pages 391–404. USENIX Association, 2010.

[146] Roberto Perdisci, Davide Ariu, and Giorgio Giacinto. Scalable Fine-grained Behavioral Clustering of HTTP-based Malware. *Comput. Networks*, 57(2): 487–500, 2013. doi: 10.1016/J.COMNET.2012.06.022. URL `https://doi.org/10.1016/j.comnet.2012.06.022`.

[147] Rafael B. Pereira, Alexandre Plastino, Bianca Zadrozny, and Luiz H. C. Merschmann. Correlation Analysis of Performance Measures for Multi-label Classification. *Information Processing & Management*, 54(3):359–369, 2018.

[148] Michal Piskozub, Fabio De Gaspari, Freddie Smith, Luigi V. Mancini, and Ivan Martinovic. MalPhase: Fine-Grained Malware Detection Using Network Flow Data. In Jiannong Cao, Man Ho Au, Zhiqiang Lin, and Moti Yung, editors, *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, pages 774–786. ACM, 2021.

[149] Michalis Polychronakis, Panayiotis Mavrommatis, and Niels Provos. Ghost turns Zombie: Exploring the Life Cycle of Web-based Malware. *LEET 2008 - 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, 2008.

[150] MalShare Project. MalShare, 2022. URL `https://malshare.com/`.

[151] J. Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1986.

[152] M. Zubair Rafique and Juan Caballero. FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors. In Salvatore J. Stolfo, Angelos Stavrou, and Charles V. Wright, editors, *Research in Attacks, Intrusions, and Defenses - 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings*, volume 8145 of *Lecture Notes in Computer Science*, pages 144–163. Springer, 2013.

[153] Jesse Read, Bernhard Pfahringer, and Geoffrey Holmes. Multi-label classification using ensembles of pruned sets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 995–1000. IEEE Computer Society, 2008.

[154] Martin Roesch. Snort: Lightweight Intrusion Detection for Networks. In David W. Parter, editor, *Proceedings of the 13th Conference on Systems Administration (LISA-99), Seattle, WA, USA, November 7-12, 1999*, pages 229–238. USENIX, 1999.

[155] Christian Rossow, Christian J. Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten van Steen, Felix C. Freiling, and Norbert Pohlmann. Sandnet: Network Traffic Analysis of Malicious Software. In Engin Kirda and Thorsten Holz, editors, *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS@EuroSys 2011, Salzburg, Austria, April 10, 2011*, pages 78–88. ACM, 2011.

[156] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. In *22nd Annual Computer Security Applications Conference (ACSAC 2006), 11-15 December 2006, Miami Beach, Florida, USA*, pages 289–300. IEEE Computer Society, 2006.

[157] Jake Ryan, Meng-Jang Lin, and Risto Miikkulainen. Intrusion Detection with Neural Networks. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]*, pages 943–949. The MIT Press, 1997.

[158] Leander Schietgat, Celine Vens, Jan Struyf, Hendrik Blockeel, Dragi Kocev, and Saso Dzeroski. Predicting Gene Function Using Hierarchical Multi-label Decision Tree Ensembles. *BMC Bioinformatics*, 11:2, 2010.

[159] Scopus. Scopus Trends, 2023. `https://bit.ly/scopus_trend`.

[160] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. Malrec: Compact Full-Trace Malware Recording for Retrospective Deep Analysis. In Cristiano Giuffrida, Sébastien Bardin, and Gregory Blanc, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018, Saclay, France, June 28-29, 2018, Proceedings*, volume 10885 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2018.

[161] Monirul Islam Sharif, Vinod Yegneswaran, Hassen Saïdi, Phillip A. Porras, and Wenke Lee. Eureka: A Framework for Enabling Static Malware Analysis. In Sushil Jajodia and Javier López, editors, *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, volume 5283 of *Lecture Notes in Computer Science*, pages 481–500. Springer, 2008.

[162] Yashovardhan Sharma, Simon Birnbach, and Ivan Martinovic. RADAR: A TTP-based Extensible, Explainable, and Effective System for Network Traffic Analysis and Malware Detection. In Aleksandra Mileva, Steffen Wendzel, and Virginia N. L. Franqueira, editors, *Proceedings of the 2023 European Interdisciplinary Cybersecurity Conference, EICC 2023, Stavanger, Norway, June 14-15, 2023*, pages 159–166. ACM, 2023.

[163] Yashovardhan Sharma, Eleonora Giunchiglia, Simon Birnbach, and Ivan Martinovic. To TTP or not to TTP?: Exploiting TTPs to Improve ML-based Malware Detection. In *IEEE International Conference on Cyber Security and Resilience, CSR 2023, Venice, Italy, July 31 - Aug. 2, 2023*, pages 8–15. IEEE, 2023.

[164] Kamran Shaukat, Suhuai Luo, Vijay Varadharajan, Ibrahim A. Hameed, and Min Xu. A Survey on Machine Learning Techniques for Cyber Security in the Last Decade. *IEEE Access*, 8:222310–222354, 2020.

[165] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Takeshi Yada. Efficient Dynamic Malware Analysis Based on Network Behavior Using Deep Learning. In *2016 IEEE Global Communications Conference, GLOBECOM 2016, Washington, DC, USA, December 4-8, 2016*, pages 1–7. IEEE, 2016.

[166] Xiaokui Shu, Frederico Araujo, Douglas Lee Schales, Marc Ph. Stoecklin, Jiyong Jang, Heqing Huang, and Josyula R. Rao. Threat Intelligence Computing. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1883–1898. ACM, 2018.

[167] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated Worm Fingerprinting. In Eric A. Brewer and Peter Chen, editors, *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 45–60. USENIX Association, 2004.

[168] Robin Sommer and Vern Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *31st IEEE Symposium on Security and Privacy, SP 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 305–316. IEEE Computer Society, 2010.

[169] Kyle Soska and Nicolas Christin. Automatically Detecting Vulnerable Websites Before They Turn Malicious. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 625–640. USENIX Association, 2014.

[170] Gary Stein, Bing Chen, Annie S. Wu, and Kien A. Hua. Decision Tree Classifier for Network Intrusion Detection with GA-based Feature Selection. In Mário Guimarães, editor, *Proceedings of the 43nd Annual Southeast Regional Conference, 2005, Kennesaw, Georgia, Alabama, USA, March 18-20, 2005, Volume 2*, pages 136–141. ACM, 2005.

[171] Yanmin Sun, Andrew K. C. Wong, and Mohamed S. Kamel. Classification of Imbalanced Data: a Review. *International Journal on Pattern Recognition and Artificial Intelligence*, 23(4):687–719, 2009.

[172] Sathya Chandran Sundaramurthy, Jacob Case, Tony Truong, Loai Zomlot, and Marcel Hoffmann. A Tale of Three Security Operation Centers. In Robert Biddle, Bill Chu, Emerson R. Murphy-Hill, and Heather Richter Lipford, editors, *Proceedings of the 2014 ACM Workshop on Security Information Workers, SIW '14, Scottsdale, Arizona, USA, November 7, 2014*, pages 43–50. ACM, 2014.

[173] Peter Szor and Peter Ferrie. Hunting for Metamorphic. In *Virus bulletin conference*. Prague, 2001.

[174] Arman Tajbakhsh, Mohammad Rahmati, and Abdolreza Mirzaei. Intrusion Detection Using Fuzzy Association Rules. *Applied Soft Computing*, 9(2): 462–469, 2009.

[175] Adane Nega Tarekegn, Mario Giacobini, and Krzysztof Michalak. A Review of Methods for Imbalanced Multi-label Classification. *Pattern Recognition*, 118:107–965, 2021.

[176] SophosLabs Research Team. Emotet Exposed: Looking Inside Highly Destructive Malware. *Network Security*, 2019(6):6–11, 2019.

[177] Bernhard Tellenbach, Martin Burkhart, Dominik Schatzmann, David Gugelmann, and Didier Sornette. Accurate Network Anomaly Classification with Generalized Entropy Metrics. *Computer Networks*, 55(15):3485–3502, 2011.

[178] Virus Total. Virustotal-free Online Virus, Malware and URL Scanner. *Online: https://www.virustotal.com/en*, 2012.

[179] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. Intrusion Detection by Machine Learning: A Review. *Expert Systems with Applications*, 36(10):11994–12000, 2009.

[180] Grigorios Tsoumakas, I. Katakis, and I. Vlahavas. Effective and Efficient Multilabel Classification in Domains with Large Number of Labels. In *Proceedings of ECML/PKDD — Workshop on Mining Multidimensional Data*, 2008.

[181] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of Machine Learning Techniques for Malware Analysis. *Computer Security*, 81:123–147, 2019.

[182] Celine Vens, Jan Struyf, Leander Schietgat, Saso Dzeroski, and Hendrik Blockeel. Decision Trees for Hierarchical Multi-label Classification. *Machine Learning*, 73(2):185–214, 2008.

[183] VirusShare.com. VirusShare, 2022. URL `https://virusshare.com/`.

[184] Gernot Vormayr, Tanja Zseby, and Joachim Fabini. Botnet Communication Patterns. *IEEE Communications Surveys & Tutorials*, 19(4):2768–2796, 2017.

[185] Hao Wang, Somesh Jha, and Vinod Ganapathy. NetSpy: Automatic Generation of Spyware Signatures for NIDS. In *22nd Annual Computer Security Applications Conference (ACSAC 2006), 11-15 December 2006, Miami Beach, Florida, USA*, pages 99–108. IEEE Computer Society, 2006.

[186] Ke Wang and Salvatore J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In Erland Jonsson, Alfonso Valdes, and Magnus Almgren, editors, *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings*, volume 3224 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 2004.

[187] Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In Diego Zamboni and Christopher Krügel, editors, *Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006, Hamburg, Germany, September 20-22, 2006, Proceedings*, volume 4219 of *Lecture Notes in Computer Science*, pages 226–248. Springer, 2006.

[188] Shanshan Wang, Zhenxiang Chen, Qiben Yan, Bo Yang, Lizhi Peng, and Zhongtian Jia. A Mobile Malware Detection Method Using Behavior Features in Network Traffic. *Journal of Network and Computer Applications*, 133:15–25, 2019.

[189] Shanshan Wang, Zhenxiang Chen, Qiben Yan, Ke Ji, Lizhi Peng, Bo Yang, and Mauro Conti. Deep and Broad URL Feature Mining for Android Malware Detection. *Information Science*, 2020.

[190] Yi-Min Wang, Roussi Roussev, Chad Verbowski, Aaron Johnson, Ming-Wei Wu, Yennun Huang, and Sy-Yen Kuo. Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management. In Lee Damon, editor, *Proceedings of the 18th Conference on Systems Administration (LISA 2004), Atlanta, USA, November 14-19, 2004*, pages 33–46. USENIX, 2004.

[191] Jonatas Wehrmann, Ricardo Cerri, and Rodrigo C. Barros. Hierarchical Multi-Label Classification Networks. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5225–5234. PMLR, 2018.

[192] Carsten Willems, Thorsten Holz, and Felix C. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security & Privacy*, 5 (2):32–39, 2007.

[193] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. A Close Look on *n*-grams in Intrusion Detection: Anomaly Detection vs. Classification. In Ahmad-Reza Sadeghi, Blaine Nelson, Christos Dimitrakakis, and Elaine Shi, editors, *AISec'13, Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*, pages 67–76. ACM, 2013.

[194] Shelly Xiaonan Wu and Wolfgang Banzhaf. The Use of Computational Intelligence in Intrusion Detection Systems: A Review. *Applied Soft Computing*, 10(1):1–35, 2010.

[195] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[196] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael S. Steinbach, David J. Hand, and Dan Steinberg.

Top 10 Algorithms in Data Mining. *Knowledge and Information Systems*, 14 (1):1–37, 2008.

[197] Jianyun Xu, Andrew H. Sung, Patrick Chavez, and Srinivas Mukkamala. Polymorphic Malicious Executable Scanner by API Sequence Analysis. In Masumi Ishikawa, Shuji Hashimoto, Marcin Paprzycki, Emilia I. Barakova, Kaori Yoshida, Mario Köppen, David W. Corne, and Ajith Abraham, editors, *4th International Conference on Hybrid Intelligent Systems (HIS 2004), 5-8 December 2004, Kitakyushu, Japan*, pages 378–383. IEEE Computer Society, 2004.

[198] Ỹanminsun, Andrew Wong, and Mohamed S. Kamel. Classification of Imbalanced Data: a Review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23:687–719, 11 2009.

[199] Ting-Fang Yen and Michael K. Reiter. Traffic Aggregation for Malware Detection. In Diego Zamboni, editor, *Detection of Intrusions and Malware, and Vulnerability Assessment, 5th International Conference, DIMVA 2008, Paris, France, July 10-11, 2008. Proceedings*, volume 5137 of *Lecture Notes in Computer Science*, pages 207–227. Springer, 2008.

[200] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xin-Zheng He. A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. *IEEE Access*, 5:21954–21961, 2017.

[201] Heng Yin, Dawn Xiaodong Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 116–127. ACM, 2007.

[202] Akira Yokoyama, Kou Ishii, Rui Tanabe, Yinmin Papa, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, Daisuke Inoue, Michael Brengel, Michael Backes, and Christian Rossow. SandPrint: Fingerprinting Malware Sandboxes to Provide Intelligence for Sandbox Evasion. In Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquín García-Alfaro, editors, *Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings*, volume 9854 of *Lecture Notes in Computer Science*, pages 165–187. Springer, 2016.

[203] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque. Random-Forests-Based Network Intrusion Detection Systems. *IEEE Transactions System, Man and Cybernetics Part C*, 38(5):649–659, 2008.

[204] Jun Zhang, Chao Chen, Yang Xiang, Wanlei Zhou, and Yong Xiang. Internet Traffic Classification by Aggregating Correlated Naive Bayes Predictions. *IEEE Transactions on Information Forensics and Security*, 8(1):5–15, 2013.

[205] Rongfeng Zheng, Jiayong Liu, Weina Niu, Liang Liu, Kai Li, and Shan Liao. Preprocessing Method for Encrypted Traffic Based on Semisupervised Clustering. *Security and Communication Networks*, 2020:8824659:1–8824659:13, 2020.