CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Flow-Based Detection of Linux Backdoor Communication

A NetFlow Based ML-Approach to Backdoor Detection in Linux Environments

Master's Thesis in Computer Science and Engineering

NAOMI ESPINOSA & LENIA MALKI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

Master's thesis 2024

Flow-Based Detection of Linux Backdoor Communication

A NetFlow Based ML-Approach to Backdoor Detection in Linux Environments

NAOMI ESPINOSA & LENIA MALKI

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Flow-Based Detection of Linux Backdoor Communication
A NetFlow Based ML-Approach to Backdoor Detection in Linux Environments
Naomi Espinosa & Lenia Malki

Examiner & Supervisor: Andrei Sabelfeld, Department of Computer Science and Engineering
Advisor: Martin Forssén, Recorded Future

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Flow-Based Detection of Linux Backdoor Communication
Naomi Espinosa & Lenia Malki
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The increasing prevalence of Linux-based systems and their susceptibility to malware attacks necessitates effective detection mechanisms for backdoor communication. This thesis explores the application of machine learning (ML) models to detect backdoor communication in Linux environments using flow-based data. Specifically, it leverages NetFlow traffic data. The study aims to determine the effectiveness of ML techniques in identifying malicious patterns associated with backdoor communication without inspecting the actual payload. Linux systems are underrepresented in existing benchmark datasets, which predominantly focus on Windows environments. To address this gap, our research trains models on flow data specific to Linux malware and environments. Through data preprocessing steps including feature mapping, aggregation, scaling, and feature selection methodologies like ANOVA F-test, models were trained and evaluated on both benign and malicious traffic datasets. The results indicate that ensemble models such as Random Forest (RF) and Extreme Gradient Boosting (XGBoost) can effectively distinguish between normal and anomalous traffic patterns, highlighting the potential of flow-based detection systems in enhancing network security. The Synthetic Minority Over-sampling Technique (SMOTE) was applied to address class imbalance, further improving the detection performance though in terms of precision. We conclude that flow-based data is a valuable tool for training models to classify malicious traffic in Linux environments. Future work will focus on acquiring or creating higher quality datasets of malicious Linux malware traffic to improve the capabilities of detection systems.

Keywords: backdoor detection, machine learning, NetFlow, Linux, malware, network security, anomaly detection, flow-based data, big data

Acknowledgements

We would like to extend our gratitude to our supervisor and examiner from Chalmers, Andrei Sabelfeld, for his invaluable guidance and support throughout our work. Additionally, we are grateful to Martin Forssén, our supervisor at Recorded Future, and the Analytics team at Recorded Future for their assistance in obtaining the data essential for this project.

Naomi Espinosa, Gothenburg, 2024-06-02
Lenia Malki, Gothenburg, 2024-06-02

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Research Aim and Question 2
1.1.1 Scope 2
1.2 Structure of Thesis 2

2 Background **5**
2.1 Network Traffic Patterns Across Operating Systems 5
2.2 NetFlow 6
2.2.1 Network Traffic Collection Tools 8
2.2.2 Triage - Sandbox Environment 8
2.3 Backdoors and C2 Communication 10
2.4 Machine Learning 11
2.4.1 ML Models 11
2.4.2 Feature Selection Methodologies 13
2.4.3 Techniques For Imbalanced Datasets 14
2.4.4 Evaluation Metrics & Validation Techniques 14
2.4.5 Stratified K-Fold Cross Validation 15
2.4.6 Limitations of Machine Learning 16

3 Related Works **19**
3.1 Flow-Based Data for Malware Traffic Detection 19
3.2 ML Approaches for Network Classification 20
3.3 Features Used in ML-Based Network Traffic Classification 21

4 Method **25**
4.1 Model Selection 25
4.2 Data Collection 26
4.2.1 Collection of Malicious Data 26
4.2.2 Collection of Benign Data 26
4.3 Data Preprocessing 27
4.4 Feature Selection 28
4.5 Model Training and Evaluation 30

5 Results **31**
5.1 Data Statistics . . . 31
5.1.1 Malicious Data Behaviour . . . 31
5.1.2 Flow Distribution Amongst Malicious Samples . . . 32
5.1.3 Mean and Modes of Features in Data . . . 32
5.1.4 Comparison of Incoming and Outgoing Packet Sizes . . . 33
5.1.5 Distribution of Protocol in Data . . . 34
5.1.6 Distribution of Flow Duration in Data . . . 35
5.1.7 Comparison of Outgoing vs. Incoming Packets . . . 35
5.2 Model Performances . . . 37
5.2.1 Average Performance Metrics Per Model . . . 37
5.2.2 ROC Curves and AUC scores . . . 38
5.2.3 FNR, FPR, Precision and Recall Curves . . . 39

6 Discussion **41**
6.1 Analysis of Model Results . . . 41
6.1.1 Feature Selection . . . 43
6.2 Identified Backdoor Communication Behaviour . . . 43
6.3 Quality of Data . . . 44
6.3.1 Period of Recorded Data . . . 45
6.3.2 Variation of Benign Data . . . 45
6.3.3 Variation of Malicious Data . . . 46
6.3.4 The Need of an ML-Based Detection Model . . . 46
6.4 Flow-Based Versus Packet-Based Data . . . 47
6.5 Practical Implications . . . 48
6.5.1 Vulnerability to Evasion Attacks . . . 48
6.5.2 Deployment and Cost . . . 49
6.5.3 Ethical Aspects . . . 50

7 Conclusion **51**

Bibliography **53**

List of Figures

2.1 NetFlow output sample. 8
2.2 Segment of a dynamic report generated for a malware sample submitted to Triage. 9
2.3 Details of Triage's mapping of dynamic analysis behaviour to known Mitre TTPs 10

4.1 Overview of method section. 25
4.2 Overview of pcap file extraction and conversion. 27
4.3 Correlation matrix for input variables. 30

5.1 Number of flows per sample ID distribution. 32
5.2 Incoming and outgoing packet size in malicious and benign data. . . . 34
5.3 Overview of protocol distribution in benign and malicious data. . . . 35
5.4 Distribution of flow-duration within interquartile range (25th to 75th percentiles). 35
5.5 Number of outgoing vs incoming packets per data point in malicious and benign data. 36
5.6 Performance metrics of models with SMOTE applied. 37
5.7 Performance metrics of models without SMOTE applied. 37
5.8 Interpolated ROC curves of models with SMOTE applied. 38
5.9 Interpolated ROC curves of models without SMOTE applied. 38
5.10 Average FNR & FPR by model, with and without SMOTE. 39
5.11 Average precision & recall by model, with and without SMOTE. . . . 39

List of Tables

2.1 NetFlow Version 5 Header Format. 7
2.2 NetFlow version 5 flow record format. 7
2.3 A subset of the nfdump toolset. 8
2.4 Confusion matrix for anomaly traffic classification. 15

3.1 Results from paper [3] detailing the results of backdoor detection using different NetFlow datasets. 20
3.2 Remaining NetFlow features grouped into 5 main categories as described in the paper [40]. 22
3.3 Summary of network features used for RAT detection of paper A [39] and paper B[41]. 23

4.1 Selection of machine learning algorithms. 25
4.2 Selection of nfdump fields before statistical feature selection. 28
4.3 Mapping of flg values to integers of base 2. 28
4.4 Mapping of protocols to integers. 28
4.5 F-statistic for each feature and P-values associated with each F-statistic rounded to three decimal points and sorted in descending order of F-statistic. 29
4.6 Original class distribution. 30
4.7 Class distribution with SMOTE applied. 30

5.1 Overview of Linux backdoor samples analysis. 31
5.2 Summary of techniques identified in Linux backdoor malware samples. 31
5.3 Mean and mode of numerical features in data. 33
5.4 Normalised mean and mode of flow durations for malicious and benign data. 33

6.1 Amount of false alarms with 2 980 741 flows daily. 48
6.2 Time elapsed for prepossessing and predictions for each model. 49

1

Introduction

In the digital age, the security of information systems is under constant threat from various types of cyber attacks, with malware being a particularly formidable threat. A backdoor is a technique used by different types of malware to establish unauthorised access to a system. This technique may be designed to achieve several malicious objectives, such as extracting sensitive data, monitoring and spying on the victim, deleting crucial data, or even introducing additional malicious software [1]. The specific goals vary according to the type of malware, which employs a backdoor technique. Once a backdoor has been deployed, it establishes communication with a Command and Control (C2) server that can send instructions or simply receive data. The consequences of such access are significant, as it allows attackers to conduct a wide array of harmful activities [2]. Detecting and neutralising this type of malicious technique is crucial for protecting the integrity, confidentiality, and operational functionality of information systems.

To effectively counter these threats, it is important to recognise patterns in malware communication, especially through the C2 channels that backdoors typically use. Understanding these patterns can lead to breakthroughs in detecting malicious activity using network data.

In the past decade, there has been substantial research interest in leveraging machine learning (ML) models for detecting and classifying anomalies in malicious network traffic and malware. Lately, this trend has been extended to include the detection of backdoors [3]. A common approach to train network detection models is to use traces of network traffic, either in packet-based or flow-based format. A typical flow-based format is NetFlow, a CISCO IOS technology that provides packet statistics as they traverse the router [4]. Common features recorded by NetFlow include bytes and packets sent and received, the duration of connections, protocol, and more. Given the nature of the aggregated data and statistics that NetFlow captures, it can be strategically leveraged to make informed decisions about the data traversing our networks, such as pinpointing instances of malicious traffic, including the C2 communication of backdoors.

Research has demonstrated that the behaviour of malware, such as Trojans, varies between different operating systems [5]. This highlights the ability of malware to exhibit distinct behaviours depending on the operating environment. To broaden the research landscape in this area, unlike many existing malware detection frameworks, we have chosen to focus on Linux-based malware. The intention of this is to balance

the research on this topic with respect to the choice of operating system for malware analysis.

The intersection of using NetFlow data to detect backdoors in Linux-based malware makes up a gap in the research domain and presents an opportunity for further exploration and development in order to address the specific challenges associated with identifying active backdoors on Linux systems using NetFlow logs.

Studying these two areas is crucial due to the widespread utilisation of NetFlow and the prevalence of Linux-based systems, particularly within corporate environments [6][7]. NetFlow, being extensively employed for network monitoring, presents a valuable avenue for enhancing backdoor detection capabilities. Although there are alternative logging data sources, it is crucial to assess the potential of NetFlow for the purpose of backdoor network communication detection in isolation before exploring hybrid approaches to accomplish this objective. Furthermore, given the significant presence of Linux machines on the internet, predominantly within corporate infrastructures, investigating the specific challenges associated with identifying active backdoors on Linux systems becomes imperative for comprehensive cybersecurity measures.

1.1 Research Aim and Question

This study explores the effectiveness of employing flow-based data for the detection of backdoor communication through machine learning methodologies. The research concentrates specifically on the utilisation of NetFlow traffic data exclusively gathered within Linux environments. In essence, this study tries to answer the question: How effective is machine learning in identifying Linux backdoor communication patterns from NetFlow traffic data?

1.1.1 Scope

The limitations of our research are as follows: Only IPv4 traffic will be used as a basis for model training and evaluation. Furthermore, only ingress network traffic is accommodated. Lastly, our research is confined to examining backdoor communication in Linux-based malware, specifically. Hence, all the data sets used in this research are derived from systems running Linux-based kernels.

1.2 Structure of Thesis

This thesis is structured into seven chapters, each designed to incrementally build upon the last, culminating in a discussion of the data quality, challenges, and practical implications of the detection of Linux backdoors using machine learning and NetFlow data. The layout is as follows:

Chapter 1: Introduction

The opening chapter provides an overview of the cyber threat landscape in corporate environments, focusing specifically on the backdoor technique employed by various malware families. It defines the problem of detecting such communications within network traffic and outlines the research objectives and limitations.

Chapter 2: Background

This chapter explores the mechanics of backdoor network communication, the role of NetFlow data in network security monitoring, and the application of machine learning within this domain. It critically reviews recent advancements and identifies gaps in current research, thereby justifying the necessity of this study.

Chapter 3: Related Work

This chapter reviews prior research relevant to our thesis, highlighting studies that may contribute to addressing the problem at hand. It also provides the foundational context upon which our methodology is built.

Chapter 4: Methodology

Here we describe the preprocessing steps required to prepare the data for machine learning analysis and explain the rationale behind the selection of specific machine learning models. This chapter also describes the metrics used to evaluate the model's performance. All steps are detailed in full, and any software dependencies are listed.

Chapter 5: Results

Here, the results of the applied models are presented. We look at the overall data as well as dive deeper into the statistics of each subset of our data and interpret it in the context of backdoor communication to understand what may have yielded our results.

Chapter 6: Discussion

This chapter discusses the thesis results in relation to the research objectives, dives into the challenges and successes encountered during the implementation of the framework, and examines how these factors influenced the outcomes.

Chapter 7: Conclusion

Concluding the thesis, we draw out the most important findings in our project and highlight the conclusions drawn when answering the research questions. Lastly, it details suggestions for improvement and future work within the research area of backdoor detection.

2

Background

This chapter provides a background essential for understanding the complexities involved in detecting malware C2 communication across different network environments. Initially, we explore the distinct behaviours of network traffic across various operating systems (OSs), to motivate the importance of Linux-based research in this domain. This sets the stage for an in-depth discussion on NetFlow, a critical tool in network monitoring, detailing its capabilities to capture and represent network traffic data effectively. Furthermore, we look into the world of ML and its capabilities in the domain of network anomaly detection. By examining the methodologies and justifications for employing ML techniques, this chapter lays the foundational knowledge necessary to understand the methodology employed in subsequent chapters.

2.1 Network Traffic Patterns Across Operating Systems

Scientific evidence suggests that network traffic does exhibit different patterns based on what OS is running on the host(s). The findings from the research conducted by Barath, in the article "Network behaviour analysis of selected operating systems" [8], support the claim that different OSs manifest unique network communication patterns. The study utilised a controlled test environment to monitor network behaviours of systems running Windows, macOS, and Linux, using the NetFlow protocol to analyse the network traffic data. It was conclusively demonstrated that these systems have distinct communication profiles with specific contacted addresses and protocols, which can be passively monitored to identify the OS type [8].

Conversely, building on the assertion that OSs exhibit distinct network traffic patterns, several studies have demonstrated the feasibility of passively identifying an OS based on its network communication. In passive OS fingerprinting, "passive" means identifying an OS by observing traffic without interaction. In the work, "A Machine-Learning-Based Tool for Passive OS Fingerprinting with TCP Variant as a Novel Feature" [9] , Hagos proposes and evaluates an advanced classification approach to passive OS fingerprinting using classical ML and deep learning techniques. Their work is looking at the network traffic at a more granular level, inspecting the packet headers of TCP traffic. However, their work emphasises the importance of unique OS-specific characteristics in network behaviour, such as the underlying

TCP variant. In contrast to the aforementioned study "Network behaviour analysis of selected operating systems", Hagos used a combination of both system-generated and user-generated network traffic in a successful attempt to passively identify the OS of network traffic, meaning that deductions of OS can be made from not only system-generated network traffic, but also in combination with user-generated traffic.

Similarly, in the study "Operating System Fingerprinting via Automated Network Traffic Analysis" [10], the authors introduce another method for OS fingerprinting using genetic algorithms and ML. Unlike Hagos, this method looks at header information from different protocols in the TCP/IP model, rather than just the TCP protocol. Thus, their approach is capable of classifying any packet using TCP/IP headers. These findings reinforce the notion that it is feasible to identify OS-specific network behaviours through passive analysis of network traffic using packet header data.

Research demonstrates that passive network data contains distinct patterns specific to each OS. Additionally, ML methods can effectively detect these patterns. This underscores the importance of recognising OS-specific differences in network traffic patterns. Moreover, this would suggest that benign and malicious traffic from hosts with different OSs would also show distinctions, highlighting the necessity of accounting for OS-specific nuances when training ML models for classifying network traffic.

2.2 NetFlow

Network traffic monitoring plays an important role in modern cybersecurity efforts, enabling the identification of potential threats and the optimisation of network performance. Two common formats for acquiring and representing network traffic are packet-based and flow-based formats. In a packet-based format, network traffic is captured and analysed at the level of individual packets. Detailed information on individual packets is thus available, including payload data. Flow-based formats provide much less granularity. Network data is aggregated into a common set of shared features such as specific source and destination IP addresses, port numbers, and protocols. Payload data is not included in flow-based data [11].

Analysing packet payloads is an impractical scenario due to the high CPU demand required for real-time analysis. Therefore, flow monitoring serves as a practical alternative for collecting network traffic for security purposes. Cisco's NetFlow protocol is an example of a flow-monitoring technology. This solution enables the collection of network traffic statistics, such as the volume of data transmitted, the frequency of communication between nodes, the types of protocols used, or other relevant network parameters, which can be used to detect network anomalies [4]. Consequently, NetFlow data can be used to assess the feasibility of identifying remote backdoor communication. Additionally, the versatility of NetFlow extends beyond security applications, finding utility in network performance optimisation and capacity planning. By providing insights into traffic patterns and resource utilisation,

NetFlow facilitates proactive network management, enabling organisations to allocate resources efficiently and maintain optimal network performance.

Cisco has released several NetFlow versions, each with its own set of features and improvements. Among these versions is NetFlow v5, which is the most commonly used [12]. Each version also comes with its own datagram format [13]. NetFlow v5 does not support IPv6 nor does it support templates for flexible flow record definition. Additionally, NetFlow v5 uses a fixed header format with predefined fields as opposed to NetFlow v9. Detailed insights into the composition of NetFlow v5 datagrams are presented in Table 2.1 and Table 2.2.

Table 2.1: NetFlow Version 5 Header Format.

Bytes	Contents	Description
0-1	version	NetFlow export format version number
2-3	count	Number of flows exported in this packet (1-30)
4-7	SysUptime	Current time in milliseconds since the export device booted
8-11	unix_secs	Current count of seconds since 0000 UTC 1970
12-15	unix_nsecs	Residual nanoseconds since 0000 UTC 1970
16-19	flow_sequence	Sequence counter of total flows seen
20	engine_type	Type of flow-switching engine
21	engine_id	Slot number of the flow-switching engine
22-23	sampling_interval	First two bits hold the sampling mode; remaining 14 bits hold the value of the sampling interval

Table 2.2: NetFlow version 5 flow record format.

Bytes	Contents	Description
0-3	srcaddr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next hop router
12-13	input	SNMP index of input interface
14-15	output	SNMP index of output interface
16-19	dPkts	Packets in the flow
20-23	dOctets	Total number of Layer 3 bytes in the packets of the flow
24-27	First	SysUptime at start of flow
28-31	Last	SysUptime at the time the last packet of the flow was received
32-33	srcport	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused (zero) bytes
37	tcp_flags	Cumulative OR of TCP flags
38	prot	IP protocol type (e.g., TCP = 6; UDP = 17)
39	tos	IP type of service (ToS)
40-41	src_as	Autonomous system number of the source, either origin or peer
42-43	dst_as	Autonomous system number of the destination, either origin or peer
44	src_mask	Source address prefix mask bits
45	dst_mask	Destination address prefix mask bits
46-47	pad2	Unused (zero) bytes

2.2.1 Network Traffic Collection Tools

Various software tools exist to capture, process and analyse network data. An example of such is nfdump which is a toolset designed for the collection and processing of, but not restricted to, NetFlow data from compatible devices [14]. It comprises several collectors, including nfcapd for NetFlow versions 1, 5, 7, and 9 as well as nfpcapd for converting PCAP data into NetFlow format. Collected data is stored in files for subsequent processing, where nfdump offers extensive capabilities for flow filtering, aggregation, and analysis. Table 2.3 shows an overview of a subset of the toolset relevant to this project. Figure 2.1 shows an output sample of an nfcapd file through the use of nfdump.

```
Date first seen         Duration         Proto      Src IP Addr:Port          Dst IP Addr:Port   Packets    Bytes Flows
2023-03-08 14:21:14.325     00:02:16.953 TCP       47.252.19.25:80    ->     10.127.0.180:60544       28     1968     1
2023-03-08 14:21:14.241     00:02:17.039 TCP       10.127.0.180:60544 ->     47.252.19.25:80          39     2738     1
2023-03-08 14:23:10.951     00:00:00.000 UDP       10.127.0.180:39610 ->       5.79.75.37:123          1       76     1
Summary: total flows: 3, total bytes: 4782, total packets: 68, avg bps: 279, avg pps: 0, avg bpp: 70
Time window: 2023-03-08 14:21:14 - 2023-03-08 14:23:31
Total flows processed: 3, passed: 3, Blocks skipped: 0, Bytes read: 228
Sys: 0.0032s User: 0.0030s Wall: 0.0010s flows/second: 3077.3 Runtime: 0.0010s
```

Figure 2.1: NetFlow output sample.

The nfpcapd tool fills a crucial gap between packet-level and flow-level data by converting raw packet data into flow records, facilitating network analysis. While packet-level data offers granular insights into individual network interactions, flow-level data provides aggregated information about communication patterns between endpoints. By capturing and converting packet data into flow records, nfpcapd enables users to bridge this divide, allowing for a more holistic understanding of network behaviour. This capability is particularly valuable in scenarios where detailed packet-level analysis is impractical due to volume or complexity, yet extensive flow-level insights are necessary for effective network monitoring, troubleshooting, and security analysis.

Tool	Description
nfdump	NetFlow display and analysing program
nfcapd	NetFlow collector daemon
nfpcapd	PCAP to NetFlow collector daemon

Table 2.3: A subset of the nfdump toolset.

2.2.2 Triage - Sandbox Environment

Malware samples can be analysed through static and dynamic analysis. Static analysis involves examining the malware's code, structure, and properties without executing it. Dynamic analysis involves executing the malware within a controlled environment, such as a virtual machine or sandbox, and observing its behaviour in real time.

Recorded Future Triage's Malware Analysis Sandbox is a tool designed to analyse and dissect potentially malicious software samples in a controlled environment [15]. This sandbox is free and publicly available. Operating within a secure and isolated

sandbox environment, it executes suspicious files or URLs and monitors their behaviour, allowing security analysts to observe and understand their actions without risking harm to their network or systems.

Each malware sample is assigned unique identifiers: *sample_id* and *task_id*. If the same malware sample undergoes analysis on multiple OSs, a distinct tuple of $sample\_id, task\_id$ is created for each analysis instance. It is possible to retrieve a generated PCAP from any analysis instance. These PCAP files are created when the sandbox environment detects network connection attempts by malware samples during dynamic analysis. Additionally, each individual instance undergoing analysis will be assigned a score ranging from 0 to 10. Instances scoring between 2 and 5 are labelled as 'Likely benign,' those scoring between 6 and 7 are labelled as 'Exhibiting suspicious behaviour,' and those with scores between 8 and 9 are labelled as 'Likely malicious.' Instances receiving a score of 10 are categorised as 'Known to be malicious'. This may happen if a malware family is detected such as in Figure 2.2 which displays a segment of the dynamic report generated for a malware sample submitted to Triage. The malware family in this example is "BPFDOOR".

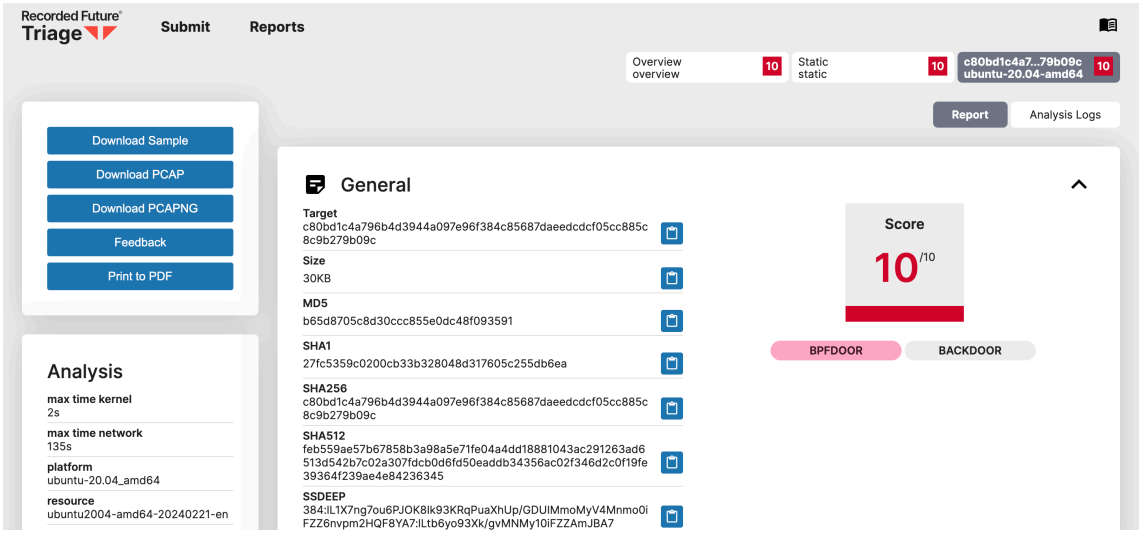

Figure 2.2: Segment of a dynamic report generated for a malware sample submitted to Triage.

To attribute a malware family or a broader malware category label to a sample ID, Triage employs several methods. All of Triage's labelling is done internally to ensure that those who submit samples cannot label them arbitrarily, preventing data poisoning through mislabelling. Triage uses signatures to match the actions or payloads of samples with known malware families. YARA, a tool for identifying and classifying malware based on textual or binary patterns, is often used for this purpose [16]. Malware payloads are identified via YARA rules crafted for specific samples. File hashes are also compared against third-party lists of known malware hashes. If a sample cannot be attributed to a family, Triage maps behaviours identified in dynamic analyses to specific Tactics, Techniques and Procedures (TTPs) in MITRE's ATT&CK Matrix [17]. Figure 2.3 depicts a sample labelled with the backdoor tag, showing the specific MITRE techniques linked to this tag and the processes run by

the sample that exhibited this behaviour. The idea is that TTPs linked to a certain category of malware result in that malware category being attributed to the sample.

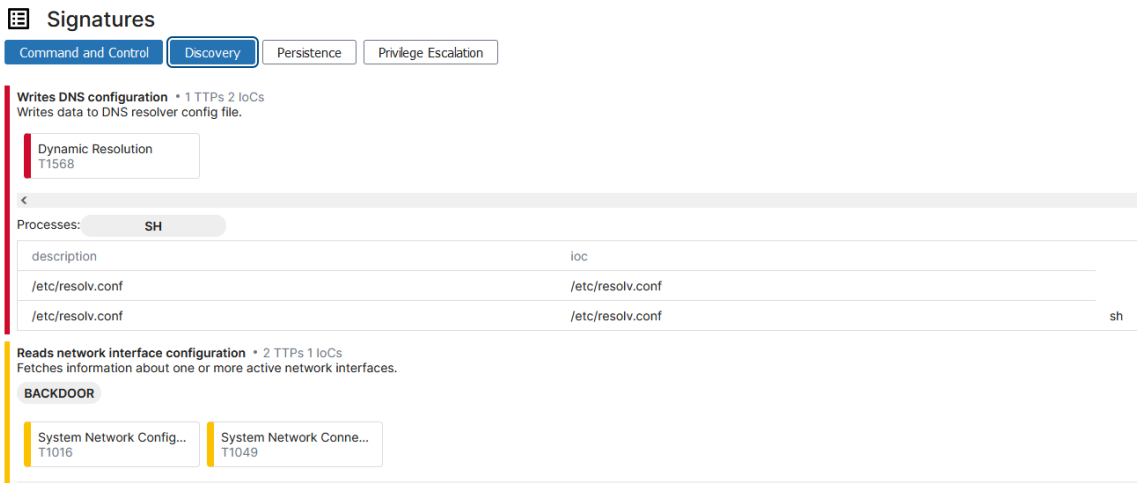

Figure 2.3: Details of Triage's mapping of dynamic analysis behaviour to known Mitre TTPs

Due to the project's time constraints, the malicious data's source was derived from Triage via the available PCAP files. An alternative methodology would involve creating a system of virtual environments, deploying a network traffic collection tool like nfcapd between designated endpoints, and executing malware samples within these environments to collect the transmitted data. However, this approach operates under the assumption that the C2 server, with which the malware communicates, is operational a circumstance that is not always guaranteed. Consequently, in the absence of a functioning C2 server, the data captured from such a setup would largely consist of Address Resolution Protocol (ARP) requests. More on how the malicious data was obtained is outlined in 4.2.1.

2.3 Backdoors and C2 Communication

A backdoor is a technique that malware can use to establish C2 communication. A backdoor is also a technique employed to establish a covert channel for data exfiltration. The MITRE ATT&CK framework, a database of tactics, and TTPs derived from real-world observations of malware and threat actors, details 18 command and control techniques and sub-techniques that adversaries may use to communicate with compromised systems [18]. Additionally, MITRE outlines 9 methods for data exfiltration, noting that those methods do not always utilise the standard C2 communication channels. Instead, malware may open separate covert channels or backdoors specifically for this purpose[19]. Since both C2 channels and potential data exfiltration channels constitute backdoors, the techniques detailed in MITREs chapters on exfiltration and C2 communication tactics are crucial for understanding what backdoors look like in the real world. Also notable is that a significant portion of the techniques and sub-techniques in this chapter of the MITRE ATT&CK framework utilises application layer protocols such as HTTPS, DNS, mail, and file transfer

protocols. In many cases, HTTPS or other encrypted channels are used for data exfiltration, and in other instances, the payload may be obfuscated, making detection techniques that attempt to examine the contents of the payload challenging.

In the context of detecting backdoor communication, certain features extracted from network traffic can be particularly revealing. These features include but are not limited to, unusual outbound traffic volumes, atypical protocol usage, and connections to suspicious IP addresses or domains. Anomalies in traffic timing, such as significant activity occurring during off-peak hours, and the presence of unrecognised or rarely used ports, can also indicate malicious activity [20].

The consistency of data packet sizes and intervals between communications might also provide clues about automated or scripted data transfers typical of backdoor exploits. NetFlow is particularly well-suited to providing these features. It captures data about ingress and egress points, traffic volume, and timing, making it an invaluable source of data for developing ML models aimed at detecting these stealthy communications. By leveraging NetFlow data, researchers can create more effective detection systems that help identify backdoor communications by analysing network flow characteristics without needing to inspect the actual payload. ML models could be used to analyse the duration and frequency of network sessions to detect patterns that deviate from the norm, which are often indicative of C2 communications or data exfiltration attempts.

2.4 Machine Learning

The subsequent sections provide an overview of ML approaches and a selection of ML algorithms relevant to our project.

ML models can be broadly categorised into three main groups: supervised, unsupervised, and semi-supervised learning. Supervised learning entails training models on labelled data, unsupervised learning uncovers patterns in unlabelled data, and semi-supervised learning combines aspects of both approaches.

Detecting anomalies or classifying instances into multiple categories can be done using ML. An anomaly is a pattern deviating from the norm. Both anomaly detection and binary classification can employ supervised or unsupervised learning, depending on data availability and problem requirements [21]. Supervised anomaly detection can be construed as a form of binary classification task, wherein the objective is to categorise instances into one of two classes: normal or anomalous. In this approach, the algorithm is trained on labelled data, explicitly identifying anomalies as the positive class (class 1) and normal instances as the negative class (class 0). As there are a vast amount of different ML algorithms to choose from, selecting the appropriate one for a given task is important.

2.4.1 ML Models

Factors such as the volume and nature of the data, including considerations of data imbalance and the potential presence of outliers, require careful consideration

when selecting ML algorithms for a specific task. Certain ML algorithms exhibit stronger robustness to outliers or possess superior scalability for handling expansive datasets. Among the array of algorithms available, a couple have demonstrated distinct advantages in this context as presented in the "Related Works" section 3.2. The rest of this subsection gives some background information on these models.

Adaptive Boosting (ADABoost): ADABoost works by iteratively training a sequence of weak learners, typically decision trees with a single split, each focusing on the mistakes of the previous one. It assigns higher weights to the instances that were misclassified in the previous round, thus forcing subsequent weak learners to concentrate more on those instances. The final prediction is made by combining the weak learners' predictions through a weighted sum, where the weights are determined by their individual performance during training. It's commonly used for classification tasks, particularly when dealing with binary classification problems. However, it can also be adapted for regression tasks [22].

Decision Trees (DT): Decision trees partition the feature space into regions, making predictions based on simple rules inferred from the training data. At each step, the algorithm selects the feature that best splits the data into homogeneous subsets with respect to the target variable. This process continues recursively until a stopping criterion is met, such as reaching a maximum depth or no further improvement in purity. Decision trees are flexible and can be used for both classification and regression tasks. They are particularly useful when the relationships between features and the target variable are non-linear or complex [23].

Extreme Gradient Boosting (XGBoost): XGBoost is an advanced implementation of gradient boosting, designed for speed and performance. Like ADABoost, it builds an ensemble of weak learners sequentially, but it differs in its optimisation objective and regularisation techniques. XGBoost employs gradient descent optimisation to minimise a differentiable loss function, using second-order derivatives for more accurate updates. It also incorporates regularisation terms to prevent overfitting, making it particularly effective for large-scale datasets. It's widely used for classification and regression tasks, especially when dealing with structured/tabular data. XGBoost is known for its high performance and is often used in competitions and production environments where speed and accuracy are crucial [22].

Gaussian Naive Bayes (GNB): This is a probabilistic classifier based on Bayes' theorem, particularly effective for classification problems with continuous features. It assumes that the features follow a normal (Gaussian) distribution and are conditionally independent given the class label. Key concepts involve calculating the posterior probability of each class by combining the prior probability of the class and the likelihood of the data given the class, derived from the Gaussian distribution. GNB is especially useful for real-time prediction due to its simplicity and efficiency, handling large datasets well even with high-dimensional data [23].

K-Nearest Neighbours (KNN): KNN works by identifying the 'k' closest data points to a given input, based on a distance metric like Euclidean distance, and then predicting the output based on the majority label (for classification) or average value (for regression) of these neighbours. KNN is particularly effective for problems

where the data distribution is unknown and requires little to no training [23].

Random Forest (RF): Random Forest is an ensemble method that constructs multiple decision trees during training and combines their predictions to improve generalisation and robustness. Each tree is built from a bootstrap sample of the training data, and at each split, a random subset of features is considered. This randomness helps decorrelate the individual trees, reducing the risk of overfitting. The final prediction is then determined by aggregating the predictions of all the trees, often through a simple majority or averaging scheme. RFs are commonly used for classification and regression tasks [23].

2.4.2 Feature Selection Methodologies

Feature selection is a critical step in the process of network traffic classification, as it helps in identifying the most relevant attributes that contribute to distinguishing different types of network traffic. Furthermore, with feature selection, one has to deal with a smaller data set, which can result in less computational power when training ML models.

When dealing with numerical input variables and categorical target variables, a suitable feature selection technique is the Analysis of Variance (ANOVA) F-test [24]. It is a type of filter method, where features are evaluated independently of the ML model. ANOVA F-test is a statistical method used to compare the means of two or more groups to determine if there are statistically significant differences among them. In the context of feature selection for ML models, ANOVA F-test helps identify the features that are most relevant for predicting the target variable.

The *SelectKBest* function in the *scikit-learn* library is a handy tool for applying filter methods in Python [25]. It operates by computing the F-statistic and associated p-values for each feature. The p-value assesses the likelihood of observing the data under the assumption that there's no real relationship between variables, essentially indicating the probability of random chance. In simpler terms, a low p-value implies that the observed relationship likely isn't due to random fluctuations whilst a high p-value suggests that the observed relationship could reasonably occur by chance. Features with extremely low p-values (close to 0) and high F-statistics are considered highly significant.

Another method for feature selection is Correlation-based Feature Selection (CFS). It is used to identify and retain the most relevant features for predicting a target variable by analysing the correlation between features. The CFS algorithm begins by evaluating the correlation between each feature and the target variable, followed by assessing the correlation among feature pairs. It then identifies a subset of features that exhibit a strong correlation with the target variable while maintaining minimal correlation with one another. This is a useful technique to use as some ML algorithms, such as those based on Naive Bayes, assume that the features are not correlated to each other [26].

2.4.3 Techniques For Imbalanced Datasets

When dealing with imbalanced datasets where one class is significantly more prevalent than the others, traditional ML algorithms may struggle to accurately predict the minority class. To address this issue, several techniques can be employed:

Over-sampling: Over-sampling involves increasing the number of instances in the minority class by randomly duplicating existing instances or generating synthetic samples. This technique helps balance the class distribution and provides more information for the model to learn from. However, it may also lead to overfitting if not carefully implemented.

Under-sampling: Under-sampling aims to reduce the number of instances in the majority class to match the minority class. This can be done by randomly removing instances from the majority class or selecting a subset of instances using various criteria. While under-sampling can help balance the dataset, it may also discard valuable information and lead to loss of predictive power.

SMOTE (Synthetic Minority Over-sampling Technique): SMOTE is a popular technique for generating synthetic samples in the minority class. It works by creating synthetic instances along the line segments joining k minority class nearest neighbours. This helps alleviate the class imbalance problem while avoiding the overfitting issues associated with simple over-sampling.

These techniques offer different approaches to handling imbalanced datasets, and the choice of method depends on the specific characteristics of the dataset and the ML algorithm being used [27].

2.4.4 Evaluation Metrics & Validation Techniques

ML models can be evaluated in various ways. The most common ways for measuring their performance include accuracy, precision, recall, F1-score, confusion matrix, and area under the ROC curve (AUC-ROC) [28].

Accuracy: Accuracy measures the proportion of correctly classified instances among all instances. It is calculated as:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Precision: Precision measures the proportion of true positive predictions among all positive predictions made by the model. It is calculated as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall (Sensitivity): Recall, also known as sensitivity or true positive rate, measures the proportion of actual positive instances that were correctly classified by the model. It is calculated as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1-score: F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is useful when classes are imbalanced. It is calculated as:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Confusion Matrix: A confusion matrix is a table that visualises the performance of a classification model. It summaries the predictions of a classifier in a tabular format, with rows representing the actual classes and columns representing the predicted classes as shown in table 2.4. The main scores in a confusion matrix include:

- True Positive (TP): Instances correctly predicted as positive.
- True Negative (TN): Instances correctly predicted as negative.
- False Positive (FP): Instances incorrectly predicted as positive (Type I error).
- False Negative (FN): Instances incorrectly predicted as negative (Type II error).

Table 2.4: Confusion matrix for anomaly traffic classification.

Actual \Predicted	**Benign**	**Malicious**
Benign	TN	FP
Malicious	FN	TP

Area Under the ROC Curve (AUC-ROC): AUC-ROC is a performance measurement for classification problems at various threshold settings. It represents the area under the receiver operating characteristic (ROC) curve, which plots the true positive rate against the false positive rate. AUC-ROC values range from 0 to 1, where a higher value indicates better performance of the model.

2.4.5 Stratified K-Fold Cross Validation

Stratified k-fold cross-validation is a variant of k-fold cross-validation (CV) that ensures each fold is representative of the entire dataset, particularly with respect to the class distribution. This method is especially beneficial for classification problems with imbalanced class distributions [29].

The process of a standard k-fold CV is as follows:

1. **Dataset Division:** The dataset is divided into k equally (or nearly equally) sized folds.
2. **Training and Validation:** For each of the k folds:

- Use $k-1$ folds as the training set.
- Use the remaining fold as the validation set.

3. **Model Training:** Train the model on the training set and evaluate it on the validation set.
4. **Performance Aggregation:** Aggregate the performance metrics across all k folds to estimate the models overall performance.

Stratified k-fold CV follows the same basic procedure as standard k-fold cross-validation with an important modification to the fold creation process:

- **Stratification of Classes:** During the division of data into k folds, each fold is ensured to have approximately the same percentage of samples of each target class as the entire dataset. For example, if the dataset comprises 70% class 0 and 30% class 1, each fold will maintain this 70/30 distribution.
- **Repeat the k-Fold Process:** Each fold is used once as the validation set while the remaining $k-1$ folds form the training set. This process is repeated k times.
- **Aggregate Performance:** The performance metrics are aggregated across all folds to provide a comprehensive performance measure.

Stratified k-fold CV ensures that each fold is representative of the overall dataset, maintaining the same distribution of classes. This is particularly important for datasets with imbalanced classes. Furthermore, it can help to avoid bias in the performance metrics that can arise if some folds are not representative of the dataset's class distribution. A trade-off to consider is that the use of any k-fold CV can be computationally expensive and time-consuming, particularly for large datasets and complex models.

2.4.6 Limitations of Machine Learning

A significant challenge with ML models is the limited availability of publicly accessible and up-to-date datasets necessary, particularly labelled ones, for training [30], [31], [32]. To solve this problem one could generate and collect new network data in a controlled environment where it is possible to separate malicious traffic from benign traffic and thus label the data. Whereas normal traffic may be easier to acquire by collecting traffic data from a real-world network context, it becomes hard to verify that there are no instances of malicious traffic in the data. Data verification directly impacts the reliability and performance of ML models in anomaly detection. Accurately verified data ensures that the models are trained on correct and representative examples, which is critical for the models to perform well in real-world scenarios. Poor data verification can lead to models that are either over-sensitive (high false positives) or under-sensitive (high false negatives), both of which are undesirable in critical applications like network security.

Another problem at hand is the choice between supervised and unsupervised learning. Unsupervised anomaly detection algorithms are more flexible when it comes to an

imbalance between anomaly and non-anomaly classes in the data and may also be fitting when the labelled data is difficult to produce with high accuracy. On the other hand, the use of unsupervised ML algorithms may also introduce a greater difficulty in terms of interpretation of results, as they operate solely based on patterns in the data without reference to labelled examples of malicious behaviour [32]. Supervised ML algorithms allow for more accurate detection and classification of anomalies though they require a large amount of labelled training data, which may not always be available in real-world scenarios. Furthermore, they may face challenges with the class imbalance problem. This problem evolves around imbalanced datasets, where you fail to capture the minority class of the dataset [33].

3

Related Works

The presence of backdoors in Linux-based systems represents a particularly insidious risk, allowing unauthorised access and control over sensitive information and resources. The detection of such backdoors is crucial for safeguarding the confidentiality, integrity, and availability of data and services. Over the years, researchers and practitioners have explored various techniques and methodologies to address this challenge, later, with a particular focus on leveraging ML algorithms and NetFlow data analysis for effective detection and mitigation. In this related works section, we dive into the existing literature on detecting malware network communication using ML and NetFlow data, examining key findings and methodologies in the field.

The first paper [1] looking into the problem of network-based backdoor detection was written by Yin Zhang from Cornell University and was published year 2000 in the 9th USENIX Security Symposium. The paper details many of the problems with detecting backdoor network communication that holds fast to this day; how to define what constitutes a backdoor and distinguishing backdoor traffic from large quantities of legitimate traffic. Zhang highlights one of the main challenges is distinguishing legitimate traffic that resembles backdoors from actual malicious backdoors. He emphasises that well-defined policies regarding normal traffic behaviour, such as the typical ports and services used, can aid in differentiating malicious and benign traffic. Zhang also notes that creating detection mechanisms or algorithms for all types of backdoors and normal traffic can be challenging, but specialised solutions can be developed for more specific network segments or familiar networks.

Naturally, the field of network security has evolved significantly since 2000, and the research field has increasingly come to be dominated by ML-based detection methods in recent years [34].

3.1 Flow-Based Data for Malware Traffic Detection

Typically, backdoor detection is done by Network Intrusion Detection Systems (NIDS) inspecting packets (or their aggregated flows) as they traverse the network. The data is processed by a detection algorithm or framework that classifies a flow or a packet as malicious or benign [34]. While packet-based data may provide more information to perform analysis on, it is less scalable and adapts poorly to implementation in

corporate infrastructures [35].

Notably, the paper titled 'NetFlow Datasets for Machine Learning-based Network Intrusion Detection Systems' [3] is especially important to our research and offered insights into using flow-based datasets for model training. The paper created NetFlow versions of four benchmark NIDS packet-based datasets: UNSW-NB15, BoT-IoT, ToN-IoT, and CSE-CIC-IDS2018. The generated NetFlow datasets were named NF-UNSW-NB15, NF-BoT-IoT, NF-ToN-IoT, and NF-CSE-CIC-IDS2018. Although many of the original datasets do contain data points labelled as a backdoor (UNSW-NB15, CSE-CIC-IDS2018), they either fail to mention the OS of the client-side susceptible to the backdoor (UNSW-NB15) [36], does not have backdoor labelled data (NF-CSE-CIC-IDS2018)[3] or, only have backdoor attacks for a Windows and macOS client-side (CSE-CIC-IDS2018) [37].

Given the potential differences in network traffic patterns among various operating systems discussed in Section 2.1, the absence of Linux network data in available benchmark datasets is problematic for our research. Ideally, backdoor traffic from Linux hosts would be needed to develop detection models that accurately can detect backdoor communications in Linux environments, to fill a critical gap in current research methodologies.

Furthermore, the study 'NetFlow Datasets for Machine Learning-based Network Intrusion Detection Systems' [3] trained models using the created datasets and provided a detailed analysis of the results. Of particular interest to our research is the detection of backdoor attacks, which are detailed in Table 3.1. The findings specifically demonstrate the effectiveness of using NetFlow data to detect backdoors in traffic originating from Windows clients/hosts. Despite varying results across the datasets, there was a notable increase in accuracy for detecting backdoor activity compared to the traditional packet-based counterparts.

Table 3.1: Results from paper [3] detailing the results of backdoor detection using different NetFlow datasets.

Dataset	**NetFlow DR**	**NetFlow F1**	**Packet-Based DR**	**Packet-Based F1**
UNSW-NB15	39.17%	0.17	13.96%	0.08
ToN-IoT	99.22%	0.98	98.05%	0.31
CSE-CIC-IDS2018	N/A	N/A	N/A	N/A
NF-UQ-NIDS	90.95%	0.92	N/A	N/A

3.2 ML Approaches for Network Classification

Network anomaly detection has been the subject of extensive research in the field of cybersecurity, as evidenced by a comprehensive survey conducted by the authors of [34]. 290 research articles published between 2000 and 2020 used ML methodologies for anomaly detection. Among these articles, the most common applications were Intrusion Detection Systems (IDSs) and network anomaly detection, comprising 68

and 66 articles, respectively. This paper highlights the use of ML algorithms as a popular and feasible method to develop anomaly detection models based on network traffic.

Nassif et al. [34] highlight that while unsupervised learning is frequently applied due to its suitability in environments with limited labelled data, the evolving complexity of anomaly detection calls for more sophisticated approaches in which supervised or semi-supervised approaches would be ideal. The authors also looked at what types of models and feature selection were suitable for different types of applications. Nassif et al. point out the growing reliance on hybrid models that combine multiple ML techniques, and ensemble methods that aggregate predictions from several models to improve accuracy and robustness. These approaches are particularly effective in handling the diverse and complex nature of modern anomalies. The critical role of feature selection and extraction is emphasised, with techniques like Principal Component Analysis (PCA) and CFS being pivotal in reducing noise and enhancing model performance. However, the selection of a suitable ML algorithm for a given task constitutes a central phase in constructing an ML model with the capability to effectively address the task at hand.

In the context of ML techniques for traffic-flow-based intrusion detection, "methods based on decision trees [...] have turned out to be the most efficient". Examples of such algorithms are RF and DT. This was concluded by the authors of [38] whom analysed various ML techniques to ascertain which ones yielded optimal traffic classification outcomes. On the other hand, algorithms such as ADABoost, GNB and KNN have repetitively shown good performance in other network classification tasks where ML has been applied [39], [34].

3.3 Features Used in ML-Based Network Traffic Classification

In a study on network anomaly detection using sampled NetFlow data, significant emphasis was placed on the optimisation of feature selection from NetFlow version 5 data. The researchers focused on the features that would potentially yield the highest information gain for anomaly detection models. This process reduced the number of features from the original twenty-four to eleven. The selected features, which can be seen in table 3.2 include both traffic descriptors and network interface data, which are crucial for analysing network flows [40].

Table 3.2: Remaining NetFlow features grouped into 5 main categories as described in the paper [40].

Retained Features
Source IP
Destination IP
Input interface
Output interface
Packets
Bytes
Source port
Destination port
Flags
Protocol
Type of Service (ToS)

Subsequently, to manage the complexity and dimensionality of the data, PCA was employed. This statistical technique was utilised to reduce the dimensionality of the feature set from eleven to five principal components. This reduction was made to preserve 95% of the variance in the data, thus maintaining the integrity and the predictive power of the models while significantly reducing computational overhead [40].

The paper [39] "An Approach to Detect Remote Access Trojan in the Early Stage of Communication" employs a unique approach to data formatting for anomaly detection, which, although not strictly adhering to standard flow-based formats like IPFIX or NetFlow, is derived from packet headers in pcap data transformed into a custom format. This custom approach to feature extraction from early-stage TCP session data is particularly relevant to our research interests, as it provides a refined method for identifying potential malicious activities through initial traffic behaviours. While the study concentrates specifically on detecting Remote Access Trojans (RATs) rather than backdoors, the nature of RAT traffic which can be considered a specific instance of backdoor traffic makes their findings relevant. The methodology focuses on capturing a combination of packet and data size metrics that reflect the typical operational patterns of RATs, which often involve significant outbound data transfer with minimal inbound communication. Find a summary of the key features used in their detection model in column A of table 3.3. The five original features (PacNum, OutByte, OutPac, InByte and InPac) were chosen based on existing works. To gain more information from these features, the authors the two additional features (O/Ipac and OB/OP) were calculated.

Additionally, the paper [41] "Optimal Remote Access Trojans Detection Based on Network Behavior" by Khin Swe Yin and May Aye Khine also contributes to this discussion by exploring the network behaviour-based detection model. This study addresses the challenge of detecting RATs in their early stages by focusing on the first twenty packets from the SYN of the TCP three-way handshake to the twentieth packet. In Table 3.3, a comparison of the features used by both papers is shown,

providing an overview of the various metrics utilised in semi-flow-based detection models for RATs. This paper [41] will be referred to as paper B in the study.

Table 3.3: Summary of network features used for RAT detection of paper A [39] and paper B[41].

Feature	**Paper A**	**Paper B**	**Description**
PacNum	X		Number of packets in the early stage of communication
Duration		X	Duration of a flow
OutByte	X	X	Total size of outbound data
OutPac	X		Number of outbound packets
InByte	X	X	Total size of inbound data
InPac	X		Number of inbound packets
O/Ipac	X	X	Ratio of outbound to inbound packets
OB/OP	X	X	Average size of outbound packets
IB/IP		X	Average size of inbound packets
OB/IB		X	Ratio of outbound data to inbound data

4

Method

The following chapter presents the methodology followed to address our research question. In the section on Model Selection, ML algorithms were chosen based on findings from related works. The Data Collection section explains the process of collecting both malicious and benign data, with specific filtering criteria detailed. Data Preprocessing outlines procedures for transforming and refining the dataset, including conversion to CSV format, flow aggregation, and feature engineering. Feature Selection utilised ANOVA F-values and p-values to select top features for analysis. Model Training and Evaluation employed stratified k-fold CV, with and without SMOTE oversampling, to evaluate each model's performance. Finally, the Limitations section addresses constraints and considerations regarding data collection and methodology.

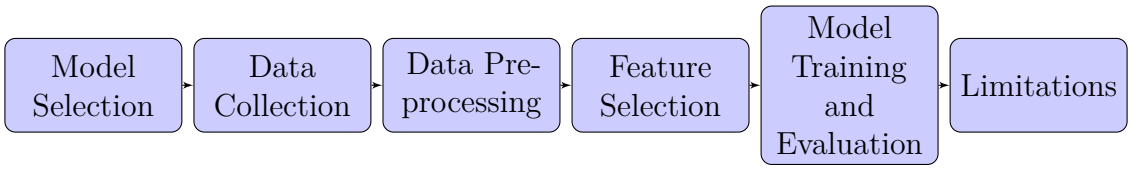

Figure 4.1: Overview of method section.

4.1 Model Selection

Based on the findings from the related works section 3.2, the ML algorithms in 4.1 were chosen for the task of binary network traffic classification.

Table 4.1: Selection of machine learning algorithms.

Machine Learning Algorithm	**Abbreviation**
Adaptive Boosting	ADABoost
Bayesian Naive Bayes	BNB
Decision Trees	DT
Extreme Gradient Boosting	XGBoost
Random Forest	RF

4.2 Data Collection

The following section explains how data was collected to train and evaluate the selection of machine learning algorithms.

4.2.1 Collection of Malicious Data

PCAP files from Triage were collected based on a set of search filters. The pseudocode in Algorithm 1 shows the criteria for extracting these PCAP files.

Algorithm 1: Pseudocode for extracting PCAP files.

```
for each sample_id, task_id in submitted_samples do
    if (OS = 'Linux' and tag = 'backdoor' and Score > 5 and
     Network_Traffic_Available = TRUE) then
        extract_pcap(sample_id, task_id);
    end
end
```

Triage's scoring parameter was used to filter out samples which did not appear to exhibit any malicious behaviour according to the platform's dynamic analysis. A score above five is considered to show suspicious and/or malicious behaviour. Essentially, all PCAP files belonging to malware samples which had been (1) successfully run on a Linux platform, (2) had been tagged with the backdoor tag, (3) had a dynamic analysis score above five and (4) had successfully been able to establish a network connection were extracted.

The SHA256 value of each obtained *sample_id* was checked. This was done to remove duplicates of the same malware sample file.

4.2.2 Collection of Benign Data

Collection of benign data must ensure the integrity of the data. One way to achieve this is to generate your own benign data, maintaining full control over the network traffic contained within. A benefit of this approach is the ability to tailor the patterns and behaviours of the normal traffic. Another way is to use real-world flow data, ensuring its integrity is not compromised. The risk of compromise in real-world data must be analysed, which can only be done with knowledge of the traffic sent over the network. This data should then undergo thorough vetting to eliminate any potential threats. An effective approach is to cross-reference the collected data against a comprehensive list of known malicious IPs and domains, ensuring the dataset remains free of compromised elements.

The benign data from this project was collected from a network segment within Recorded Future's corporate infrastructure. This network segment is specifically designated for calling external APIs and downloading packages from external repositories, implying that the benign data should be communicating with predefined trusted sources. However, there remains a small risk that these sources could be

compromised, potentially introducing vulnerabilities into the systems they interact with.

To mitigate this risk and ensure the benign nature of the data, any potentially malicious data points were filtered out by comparing them against Recorded Future's extensive IP risk list. Recorded Future generates risk scores for IPs by combining intelligence gathered from their automatic analysis of unstructured text, integrating threat intelligence from multiple sources such as threat feeds, security reports, and dark web monitoring. These sources provide a comprehensive overview of risky IP addresses, making the list reliable for identifying and filtering out potential threats [43].

4.3 Data Preprocessing

This section outlines the procedures undertaken to prepare and refine the dataset for subsequent model training.

The extracted PCAP files from Triage were converted into nfcapd files using nfpcapd. The nfcapd files were further transformed into CSV files, using nfdump, while simultaneously aggregating them into bidirectional flows. Aggregation occurs at the connection level, where the 5-tuple protocol (comprising protocol type, source IP, destination IP, source port, and destination port) is considered. This helps reduce the size of the data. An overview of the extraction and conversion process is shown in figure 4.2

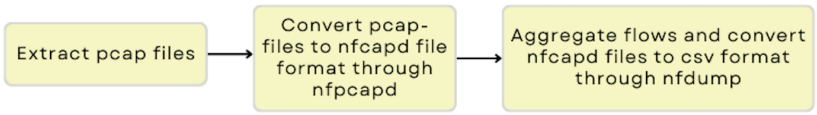

Figure 4.2: Overview of pcap file extraction and conversion.

To mitigate potential bias towards specific nodes, both source and destination IP addresses, along with source and destination ports, were deliberately omitted from the dataset. Instances of network flows associated with malicious activity were designated with the label "1", while those indicative of benign traffic were labelled as "0". Furthermore, four supplementary fields were generated and subsequently incorporated into the dataset. The twelve attributes, selected prior to any statistical feature selection process, are enumerated in Table 4.2. Furthermore, any NaN values were dropped and Inf values were replaced by 0.

Table 4.2: Selection of nfdump fields before statistical feature selection.

	Field	Description
1	td	Duration of the flow in seconds and milliseconds.
2	pr	Protocol used in the connection.
3	flg	TCP flags ORed of the connection.
4	ipkt	Input Packets
5	opkt	Output Packets
6	ibyt	Input Bytes
7	obyt	Output Bytes
8	IbytByIpkt	Number of input bytes by input packets
9	ObytByOpkt	Number of output bytes by output packets
10	ObytByIbyt	Number of output bytes by input bytes
11	OpktByIpkt	Number of output packets by input packets

The *flg* and *pr* fields were mapped to integers. Each flag is assigned a unique numerical value that corresponds to a power of 2 as shown in table 4.3. The empty string is mapped to 0, indicating no flags present. The sum of *flg* values thus represents a unique combination of flags being activated. In the mapping provided for protocols, each protocol is assigned a unique numerical value as shown in table 4.4. The mapping of the *flg* and *pr* fields was based on the documentation provided in the manual pages of NFDUMP [14].

Table 4.3: Mapping of flg values to integers of base 2.

Flag	X	U	P	R	F	S	A
Mapping	63	32	16	8	4	2	1

Table 4.4: Mapping of protocols to integers.

Protocol	RSVP	AH	ESP	GRE	ICMP	UDP	TCP
Mapping	6	5	4	3	2	1	0

The data was scaled to improve the stability and generalisation performance of the machine learning models. Two scaling methods were considered: StandardScaler and RobustScaler which are both provided by *scikit-learn*. The choice between these methods was determined based on the characteristics of the data and the desired robustness of the model. RobustScaler, chosen for its ability to handle outliers effectively, was applied to the dataset to mitigate the impact of extreme values on feature scaling. By transforming the data using RobustScaler, the features were normalised using the interquartile range, reducing the influence of outliers and enhancing the model's resilience to variations in the dataset.

4.4 Feature Selection

Feature selection was performed using the *SelectKBest* function from the *scikit-learn* library. The selection of SelectKBest with ANOVA is grounded in its ability to handle high-dimensional data and its effectiveness in identifying features that contribute

most to the variance between classes, as discussed in section 4.4 . This method simplifies the model and enhances performance by reducing the dimensionality of the input data.

The ANOVA F-values between labels and features, along with their respective p-values, as detailed in Table 4.5. Subsequently, the top six features were selected for further analysis. These features include: *pr, flg, ObytByOpkt, OpktByIpkt, OpktByIpkt*, and *td*.

Table 4.5: F-statistic for each feature and P-values associated with each F-statistic rounded to three decimal points and sorted in descending order of F-statistic.

Feature	F-statistic	P-value
pr	288113.664	0.000
flg	11972.300	0.000
ObytByOpkt	97.611	0.000
IbytByIpkt	77.446	0.000
OpktByIpkt	49.140	0.000
td	5.605	0.018
ibyt	1.040	0.308
obyt	0.961	0.327
ipkt	0.856	0.355
ObytByIbyt	0.479	0.489
opkt	0.082	0.775

The top features display the greatest difference in variance when comparing benign data with malicious data. In terms of model training and classification ability, a higher F-value indicates a greater ability to distinguish between classes, as it shows a significant variance between groups. The p-values associated with these F-values further support their significance, as they are all very low. A low p-value, typically less than **0.05**, indicates that the observed variances are highly unlikely to have occurred by chance.

The top six features were chosen because they exhibited the highest F-values, making them the most informative for our classification model. Specifically, features *pr, flg, ObytByOpkt, IbytByIpkt, OpktByIpkt* had particularly high F-values, all with corresponding p-values of 0.000, indicating strong statistical significance. The sixth feature, *td*, while having a slightly higher p-value of 0.018, still met the threshold for statistical significance and was included because it was seen as a telling and informative feature regarding malicious activity.

In order to not include features which were highly correlated with each other, CFS was also implemented. Highly correlated pairs of features could lead to overfitting of models due to features providing similar data to the model. Furthermore, some of the selected models assume no correlation between features and thus, must be removed. As no features were highly correlated with each other (see Figure 4.3), no further features were removed.

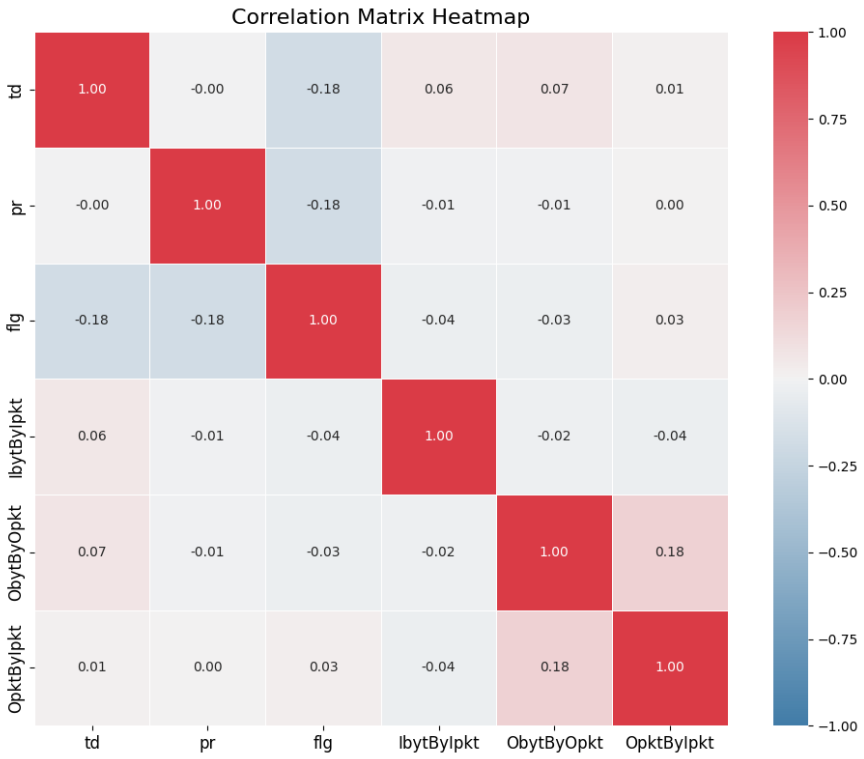

Figure 4.3: Correlation matrix for input variables.

4.5 Model Training and Evaluation

All models were trained and evaluated using stratified k-fold CV which was $k = 5$. These models underwent training and evaluation utilising varying proportions of the original dataset. In the first strategy, SMOTE was employed to oversample the minority class, this being the malicious class, by synthetically generating new samples. This strategy aimed to establish a ratio of benign to malicious samples approximately equivalent to $80 : 20$. The second approach refrained from any form of resampling. Tables 4.6 and 4.7 show the different class distributions in the original data set (after prepossessing) and when SMOTE has been applied. This approximate $80 : 20$ ratio was chosen as opposed to a $50 : 50$ ratio because, in real-life scenarios, it is not realistic for benign and malicious data to occur in equal ratios.

Performances of each fold for each model were collected and the average accuracy across all folds was computed. The average AUC was used along with interpolation for all FPRs and TPRs in order to plot an average ROC curve.

Table 4.6: Original class distribution.

Label	Count	Percentage
0	2980741	99.975
1	740	0.025

Table 4.7: Class distribution with SMOTE applied.

Class	Count	Percentage
0	2384593	83.46
1	476918	16.54

5

Results

5.1 Data Statistics

The following section presents statistics on the data which was used to train and evaluate the machine learning models.

5.1.1 Malicious Data Behaviour

This project's malicious data focused on 43 Linux backdoor samples, and 23 of these samples had an unknown malware family, however, they were labelled as a backdoor thanks to the activity identified in Triage's sandbox. 20 of the samples were attributed to backdoor malware families Iptablez, BPFDoor, XZutil, and Metasploit backdoors. See the distribution amongst our samples in table 5.1.

Table 5.1: Overview of Linux backdoor samples analysis.

Description	**Count**
Total Linux Backdoor Samples Analysed	43
Labels from Specific TTPs	23/43
Samples Attributed to Malware Families	20/43

A detailed summary of the key Mitre techniques and their occurrences within the collected samples are presented in table 5.2 below.

Table 5.2: Summary of techniques identified in Linux backdoor malware samples.

Technique	**Description**	**Count out of 43**
T1016	System Network Configuration Discovery	38
T1049	System Network Connections Discovery	26
T1568	Dynamic Resolution	14
T1053	Scheduled Task/Job	12
T1574	Hijack Execution Flow	11
T1082	System Information Discovery	29
T1497	Virtualisation/Sandbox Evasion	5
T1562	Impair Defences	4

One of the notable techniques employed by malware samples in our data is Dynamic Resolution (T1568), which complicates the tracking of C2 infrastructure by dynamically resolving domain names. Dynamic DNS (DDNS) is a method used within this technique where domain names are frequently updated with new IP addresses. This approach allows the malware to maintain consistent contact with its C&C servers even if the IP addresses change, making it more resilient against takedowns and blocking efforts [44].

Furthermore, we see techniques aimed at virtualisation and sandbox evasion (T1497) and hijacking execution flow (T1574) to modify how programs execute, redirecting execution to attacker-controlled code. Lastly, a notable trend is the network and system discovery techniques utilised for identifying active network connections, listening ports and network configurations (T1049, T1016).

5.1.2 Flow Distribution Amongst Malicious Samples

Figure 5.1 illustrates the distribution of flow occurrences per individual malware sample. Flows belonging to the same malware sample but a different behavioural, i.e. a different Linux platform, have all been aggregated.

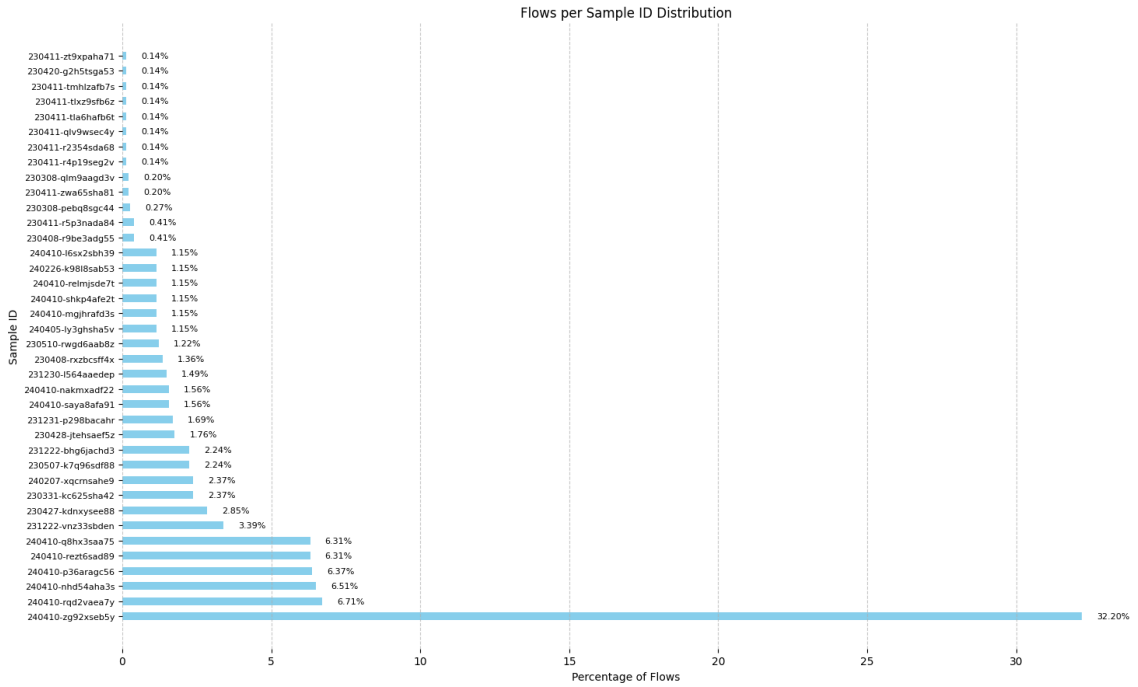

Figure 5.1: Number of flows per sample ID distribution.

5.1.3 Mean and Modes of Features in Data

Table 5.3 provides an overview of the mean and median values for key features, which include total duration (td), outbound bytes (obyt), inbound bytes (ibyt), outbound packets (opkt), inbound packets (ipkt), the ratio of inbound bytes to inbound packets

(IbytByIpkt), ratio of outbound bytes to outbound packets (ObytByOpkt), and the ratio of outbound packets to inbound packets (OpktByIpkt).

These statistics are critical for understanding the underlying patterns of network traffic associated with both benign and malicious flows. The mean and median values help highlight the typical behaviour observed in the data, whereas discrepancies between these measures can indicate the presence of outliers or skewed data distributions. Throughout the results section, we will refer back to these statistics to make comparisons on trends shown in other figures.

Table 5.3: Mean and mode of numerical features in data.

Feature	Mean Malicious	Median Malicious	Mean Benign	Median Benign
td	8304.74	79.00	14092.85	112.00
obyt	187452.07	195.50	1220782.49	6150.00
ibyt	2881.24	164.00	99423.22	2763.00
opkt	137.99	1.00	177.81	10.00
ipkt	41.52	2.00	155.64	10.00
IbytByIpkt	91.69	75.00	354.39	219.60
ObytByOpkt	354.02	170.00	881.23	648.56
OpktByIpkt	1.06	1.00	0.94	0.91

In reviewing the non-normalised mean and median values for flow duration (td) presented in Table 5.3, the different durations for which benign and malicious data were captured must be considered. These initial measurements might not be directly comparable, as the length of the observation periods significantly varies between the two datasets. However, by normalising these flow durations, details of which are provided in Table 5.4, a more accurate understanding of the data is gained. The normalised durations show that, when adjusted for the length of their respective capture periods, malicious activities engage in sustained, longer-duration flows to a much greater extent than benign activities.

Table 5.4: Normalised mean and mode of flow durations for malicious and benign data.

Type	**Normalised Mean (ms)**	**Normalised Mode (ms)**
Malicious	2.306	0.022
Benign	0.163	0.001

5.1.4 Comparison of Incoming and Outgoing Packet Sizes

Figure 5.2 reveals consistent trends across both datasets. It is observed that, on average, the size of outgoing packets surpasses that of incoming packets for both benign and malicious data. This indicates a general pattern where outgoing packets contain more data than incoming ones. Notably, benign data typically features larger

packets compared to malicious data, suggesting differences in data transmission behaviours between the two types.

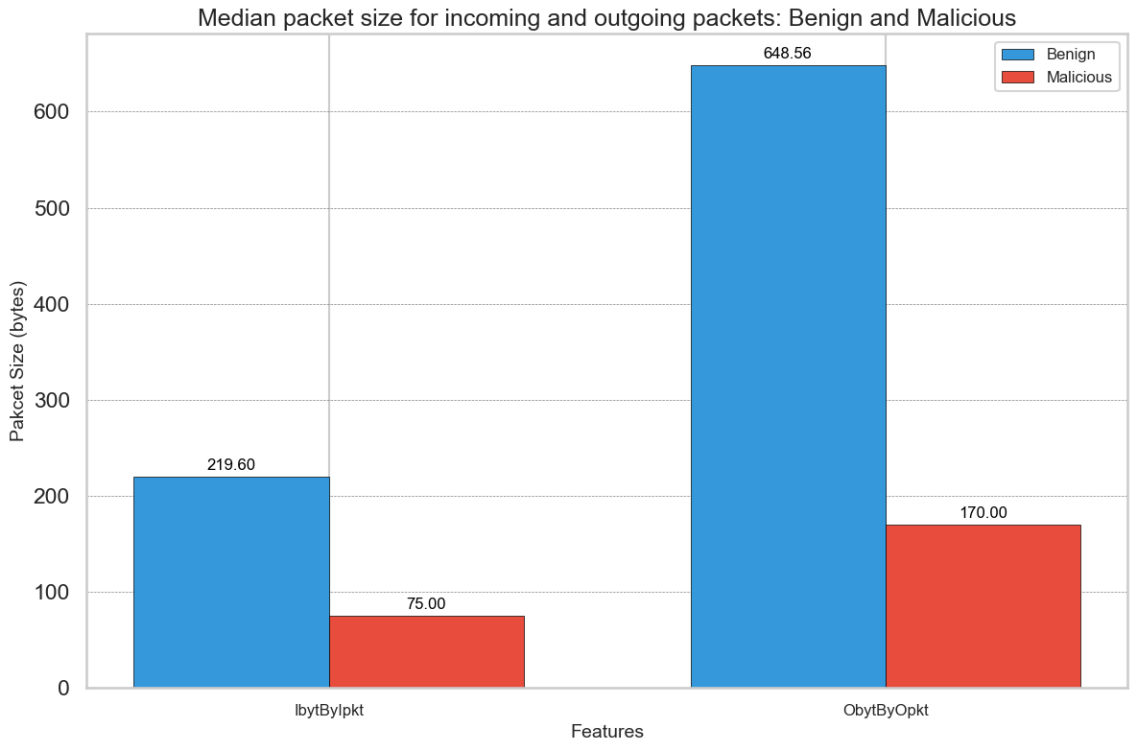

Figure 5.2: Incoming and outgoing packet size in malicious and benign data.

5.1.5 Distribution of Protocol in Data

The proportion of protocols in the data is illustrated by Figure 5.3. It is observed that the malicious data comprised approximately equal parts of the transport layer protocols, TCP and UDP, whereas the benign data consisted almost entirely (99.9%) of TCP flows. This distinct contrast in protocol distribution, with clear homogeneity in the benign dataset against the variance in the malicious data, significantly enhances the protocol feature's impact as a determinant in model classification decisions. This difference in variance is a key contributor to the high F-statistic for the protocol feature observed in the ANOVA feature selection, as detailed in Section 4.4 and shown in Table 4.5.

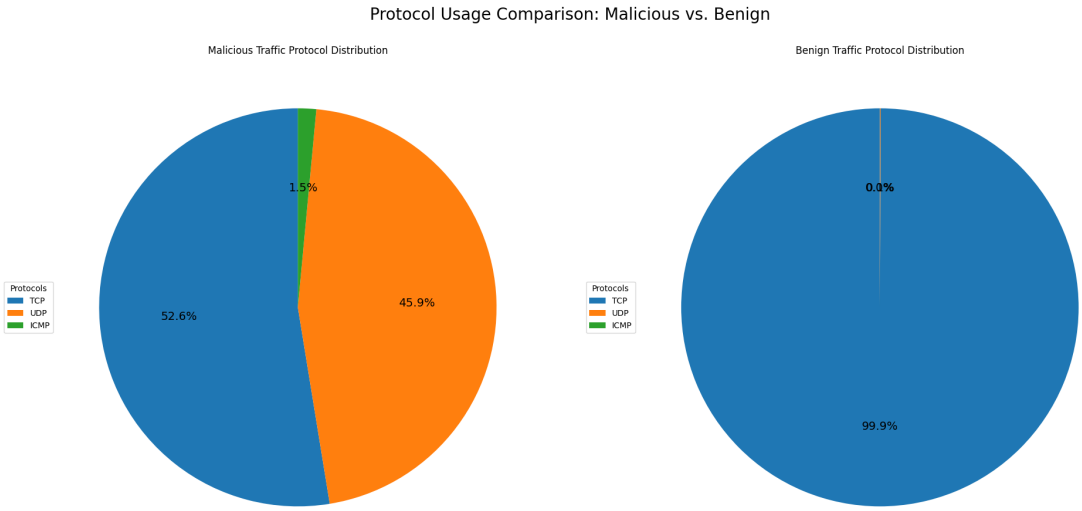

Figure 5.3: Overview of protocol distribution in benign and malicious data.

5.1.6 Distribution of Flow Duration in Data

Figure 5.4 presents a density plot comparing flow durations in malicious and benign data. The plot focuses on the interquartile range, which encompasses the middle 50% of each dataset. Examining the axes, we observe significant differences in flow durations within this range. Specifically, all flow durations in the interquartile range for benign data are below 600 ms, whereas those for malicious data extend up to approximately 3000 ms. This indicates that the range of flow duration for malicious data within this interval is about five times greater than that for benign data, suggesting more variability in the duration of malicious flows.

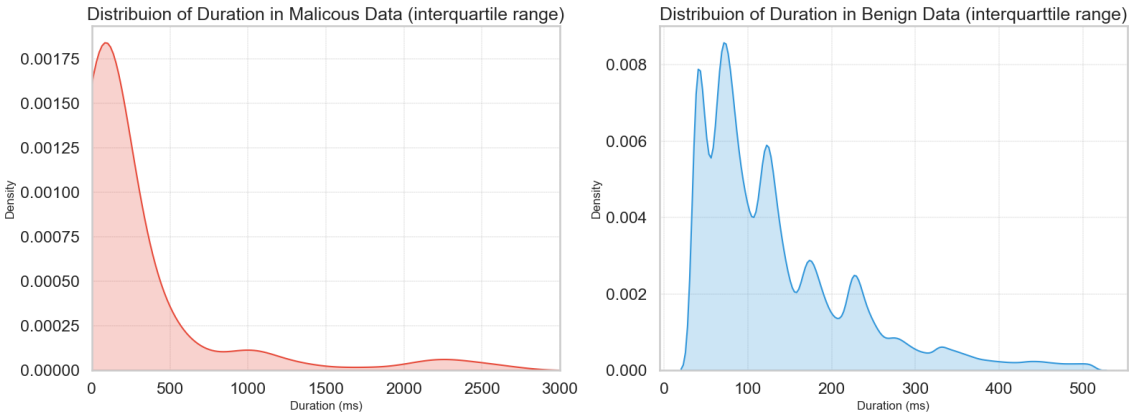

Figure 5.4: Distribution of flow-duration within interquartile range (25th to 75th percentiles).

5.1.7 Comparison of Outgoing vs. Incoming Packets

Analysis of the trend lines depicted in the graph (Figure 5.2) shows that the slope for the malicious data is noticeably steeper compared to that of the benign data.

This steeper slope suggests a more pronounced increase in outgoing packets relative to incoming packets in the malicious dataset. Such a trend indicates that malicious communications send out more outgoing packets than incoming packets, to a greater extent than observed in the benign data. Supporting this observation, data from Table 5.3 shows that the 'OpktByIpkt' feature indicating the ratio of outgoing to incoming packets has a higher median and mode in the malicious dataset than in the benign dataset.

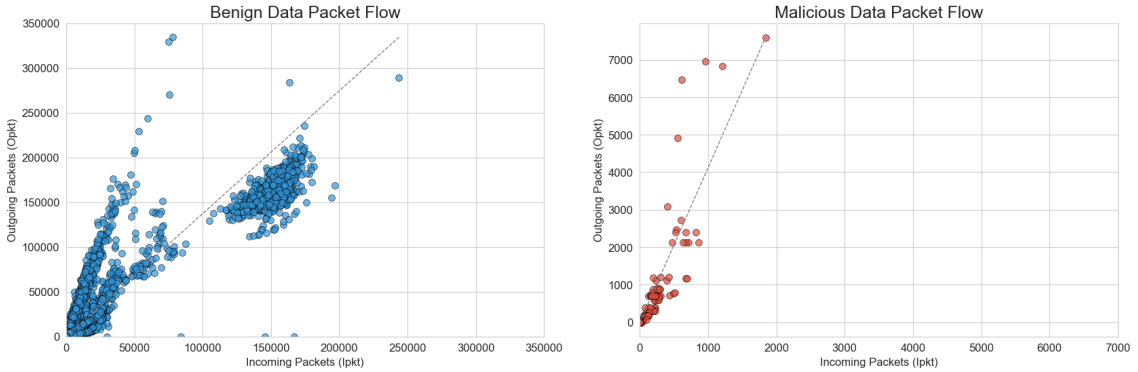

Figure 5.5: Number of outgoing vs incoming packets per data point in malicious and benign data.

5.2 Model Performances

The following section presents the results of the evaluation metrics for each model. These evaluation metrics are True Negatives (TN), False Positives (FP), False Negatives (FN), False Negative Rate (FNR), False Positive Rate (FPR), Precision, Recall as well as ROC curve (receiver operating characteristic curve) and corresponding Area under the ROC Curve (AUC). It is important to point out that the results presented in tables 5.6 and 5.7 are averages based on stratified K-Fold CV with five folds. Furthermore, the ROC curves presented in figures 5.8 and 5.9 have been interpolated for each model and the AUC represents the average AUC score of all five folds per model.

5.2.1 Average Performance Metrics Per Model

Figure 5.6: Performance metrics of models with SMOTE applied.

Model	TN	FP	FN	TP	FNR	FPR	Precision	Recall
ADABoost	595991	157	20	128	0.135135	0.000263	0.449123	0.864865
DT	595989	159	20	128	0.135135	0.000267	0.445993	0.864865
GNB	590090	6058	49	99	0.331081	0.010162	0.016079	0.668919
KNN	595839	310	21	127	0.141892	0.000520	0.290618	0.858108
RF	596038	110	19	129	0.128378	0.000185	0.539749	0.871622
XGBoost	593198	2951	19	129	0.128378	0.004950	0.041883	0.871622

Figure 5.7: Performance metrics of models without SMOTE applied.

Model	TN	FP	FN	TP	FNR	FPR	Precision	Recall
ADABoost	596132	17	29	119	0.195946	0.000029	0.875000	0.804054
DT	596126	22	30	118	0.202703	0.000037	0.842857	0.797297
GNB	590497	5652	56	92	0.378378	0.009481	0.016017	0.621622
KNN	596086	62	50	98	0.337838	0.000104	0.612500	0.662162
RF	596144	4	35	113	0.236486	0.000007	0.965812	0.763514
XGBoost	596147	1	81	67	0.547297	0.000002	0.985294	0.452703

5.2.2 ROC Curves and AUC scores

Looking solely at the ROC curves presented in figure 5.8 and 5.9, all of the models are performing better than random guess (AUC = 0.50). Furthermore, all models have higher AUC scores when SMOTE is applied to address the imbalanced dataset.

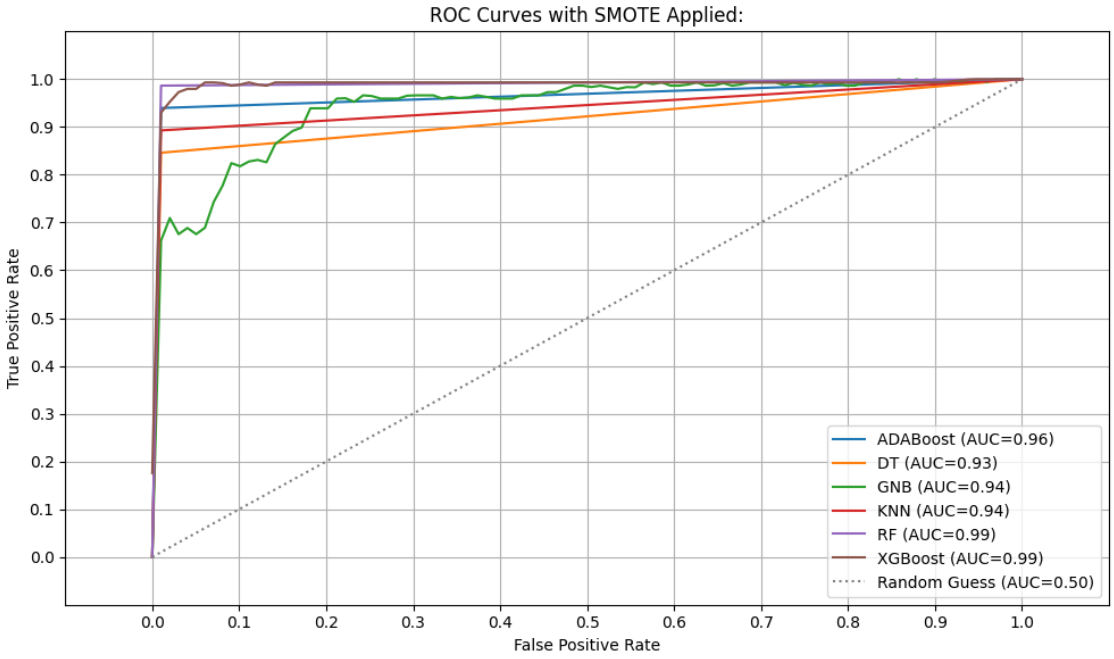

Figure 5.8: Interpolated ROC curves of models with SMOTE applied.

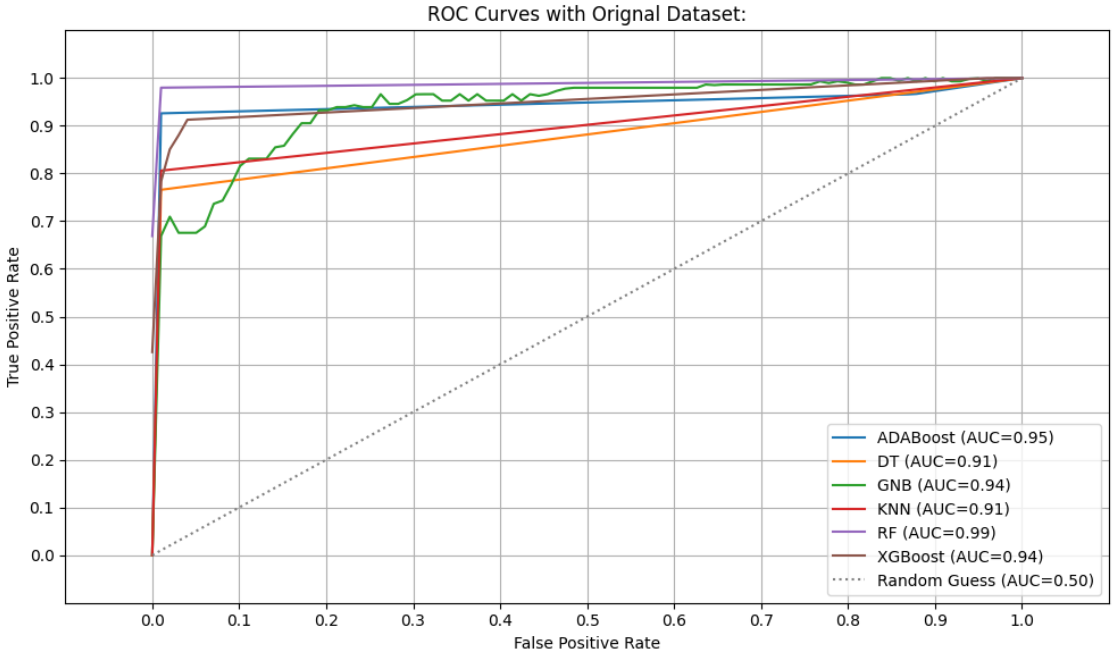

Figure 5.9: Interpolated ROC curves of models without SMOTE applied.

5.2.3 FNR, FPR, Precision and Recall Curves

In the following two figures, each model's FNR, FPR, precision and recall scores have been plotted for the two cases when SMOTE has been applied and when only the original data set has been used. The GNB classifier stands out among the models for its lack of precision and high FNR. In terms of recall, the scores are relatively similar among all models except for XGBoost which falls behind on the original data set. FPRs are relatively low for all models.

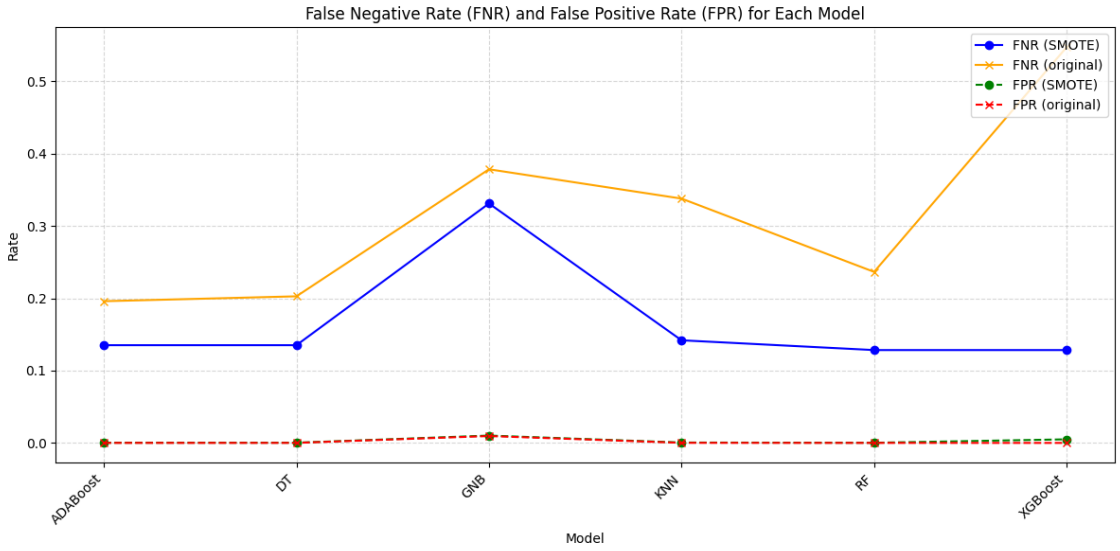

Figure 5.10: Average FNR & FPR by model, with and without SMOTE.

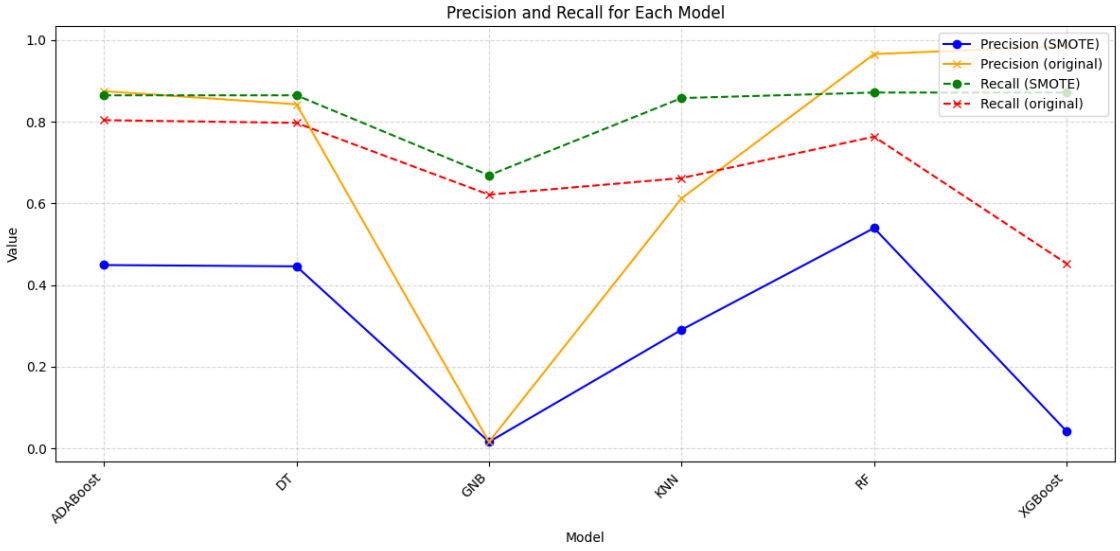

Figure 5.11: Average precision & recall by model, with and without SMOTE.

6

Discussion

This chapter analyses the performance of various ML models in classifying network traffic as benign or malicious using NetFlow data. Stratified k-fold cross-validation and SMOTE were employed to address data imbalance, revealing significant variations in model performance, particularly in precision and recall.

We discuss the impact of SMOTE on class distribution and model accuracy. Ensemble methods like Random Forest and XGBoost showed superior performance, especially in handling imbalanced datasets.

Feature selection was performed using ANOVA F-values and CFS to enhance model interpretability and performance. We highlight the challenges posed by feature correlation, particularly for models assuming feature independence, such as Gaussian Naive Bayes.

The implications of our findings are discussed in the context of practical network traffic classification, emphasising the effectiveness of the best-performing models in real-world scenarios. This chapter provides a concise evaluation of model strengths and weaknesses, guiding future improvements in cybersecurity measures.

6.1 Analysis of Model Results

This section evaluates the performance of the ML models in classifying network traffic as either benign or malicious using NetFlow data. Stratified k-fold CV was employed to ensure robust evaluation, and SMOTE was applied to address the class imbalance. The results demonstrate significant variations in model performance, particularly in precision and recall when trained on original versus SMOTE-enhanced datasets.

Consideration of the significant variance in class distribution across scenarios is imperative. Table 4.7 illustrates the class distribution post-application of SMOTE. When dealing with imbalanced datasets, traditional evaluation metrics such as accuracy can be misleading because they do not account for the imbalance between classes. Therefore, it's essential to use evaluation metrics that provide a more nuanced view of model performance, particularly on the minority class. The two metrics precision and recall have thus been used to evaluate the models, focusing on the minority class.

Random Forest (RF): RF showed robust performance due to its ensemble nature, reducing overfitting and increasing stability. The model's high AUC and precision on the original dataset highlight its effectiveness. However, SMOTE reduced precision significantly, likely due to the introduction of synthetic samples that added noise, albeit improving recall and reducing the FNR. The increase in FPR after SMOTE indicates some benign flows were misclassified as malicious.

ADABoost: ADABoost's boosting technique, which focuses on difficult-to-classify instances, demonstrated sensitivity to class imbalance. The increase in recall with SMOTE indicates a positive effect from additional minority samples, but the reduced precision and increased FPR suggest potential overfitting to these synthetic instances.

Decision Tree (DT): DTs are prone to overfitting, especially with imbalanced datasets. SMOTE improved recall by providing more minority class samples but decreased precision due to less informative synthetic samples. The increase in FPR post-SMOTE indicates a trade-off between correctly identifying malicious flows and misclassifying benign ones.

Gaussian Naive Bayes (GNB): GNB assumes normality and feature independence, which might not hold for network traffic data. Its poor performance and negligible improvement with SMOTE highlight the limitations of these assumptions in handling imbalanced data. The slight improvement in recall and a corresponding decrease in FNR came at the cost of a higher FPR.

K-Nearest Neighbours (KNN): KNN's instance-based learning is sensitive to class imbalance. SMOTE significantly increased recall but decreased precision, indicating that synthetic samples helped cover the minority class but introduced noise. The lower FNR post-SMOTE suggests better identification of malicious flows, though the increased FPR indicates more benign flows were incorrectly classified as malicious.

XGBoost: XGBoost's gradient boosting technique and regularisation parameters make it robust to different data distributions. SMOTE substantially improved recall and reduced FNR, but drastically reduced precision, suggesting overfitting to synthetic samples and a significant increase in FPR.

In conclusion, ensemble methods such as RF, XGBoost and ADABoost demonstrate superior performance in both cases where resampling was applied and not. This is consistent with the finding from related works. In practical network traffic classification scenarios, the distribution of malicious and benign data instances often exhibits a significant class imbalance. As such, the efficacy of models in handling imbalanced datasets and discerning nonlinear relationships within the data assumes crucial importance. Models demonstrating adeptness in accommodating class imbalance and capturing intricate nonlinear dependencies offer a more viable solution for accurate network traffic classification.

It is important to consider that there are specific situations where applying SMOTE not be appropriate. Its applicability is contingent upon several factors: it may cause overfitting on small datasets, exacerbate noise issues, become less effective in high-

dimensional data, fail to represent complex data distributions accurately, require significant computational resources, prove impractical for real-time applications, and mislead models when minority class outliers are present. These considerations underscore the importance of judiciously assessing the suitability of SMOTE for a given dataset and problem context.

6.1.1 Feature Selection

The importance of feature selection has previously been mentioned in section 2.4.2. Models such as RFs and DTs are less dependent on prior feature selection implementations. While prior feature selection can still improve the performance of DTs and RFs by reducing noise and improving interpretability, these models are inherently less dependent on it due to their internal mechanisms for handling features. This makes them particularly useful in situations where automated feature selection is challenging. However, models based on Naive Bayes's theorem, such as GNB, are sensitive to correlated features. This sensitivity arises because the Naive Bayes algorithm assumes that the features are conditionally independent given the class label, an assumption that rarely holds in real-world datasets. This is a possible reason why GNB is performing less compared to the other models.

Some works, as mentioned in 3.2, have opted to include features such as source and destination addresses, as well as source and destination ports, as part of their feature set. However, we dropped these features before applying any statistical feature selection. This decision is supported by several compelling reasons that underscore the effectiveness of excluding addresses and ports in network traffic classification models. Relying on addresses and ports for classification may introduce noise and reduce the generalisation capability of the model. By excluding these features, the model becomes more robust to changes in network topology and configuration, enhancing its stability and performance across different network environments. Secondly, addresses and ports may not always be indicative of malicious activity or meaningful patterns in network traffic. While certain IP addresses or port numbers may be associated with known malicious entities or services, relying solely on this information for classification can lead to false positives or miss important anomalies.

In our case, ANOVA F-values were used for feature selection because this lightweight, filter-based approach preserves interpretability, unlike PCA. ANOVA is particularly beneficial as it evaluates the significance of each feature in relation to the target variable, making it easier to understand which features contribute most to the model's predictions. However, as it does not accommodate the correlation between features themselves, CFS was used as a complementary method.

6.2 Identified Backdoor Communication Behaviour

The insights from the data visualisations and statistics in Section 5.1 draw a picture of the behavioural differences between benign and malicious network flows. Specif-

ically, the longer flow durations and larger sizes of outgoing packets compared to incoming ones in malicious data align with backdoor behaviours designed to maintain persistent connections and facilitate data exfiltration.

The significantly steeper slope observed for outgoing versus incoming packets in malicious data, as illustrated in Figure 5.2, supports the hypothesis that malicious entities engage in sending data rather than receiving it. This critical observation not only validates feature selection strategies for ML models but also underscores the importance of incorporating metrics such as the 'OutpktByIpkt' ratio - a key indicator of malicious activity, as validated by several studies discussed in Section 3.3.

Looking at the techniques utilised by malicious samples in our data, as presented in Section 5.1.1, and Table 5.2 we can make some interesting inferences. The utilisation of Dynamic Resolution (T1568) underscores the increasing sophistication of C2 communication strategies. By leveraging Dynamic DNS (DDNS), malware can ensure uninterrupted contact with C2 servers despite IP address changes, thus enhancing their resilience against conventional IP blocking and takedown efforts [44]. This adaptability complicates efforts to track and dismantle malicious infrastructure, underscoring the need for detection mechanisms that go beyond blocklist IPs to analyse patterns in network traffic as a way to counteract threat actors' evasion strategies.

Additionally, the deployment of virtualisation and sandbox evasion techniques (T1497) [45] and execution flow hijacking (T1574) [46] signifies a deliberate effort by attackers to evade detection and maintain control over infected systems. The prevalent use of network and system discovery techniques (T1049, T1016) further illustrates the comprehensive approach malware authors take to map out and exploit network environments, facilitating lateral movement and deeper infiltration.

These insights into the sampled malware's operational tactics and behavioural patterns reveal the complexity and sophistication of modern malware strategies. The demonstrated adaptability in communication methods and evasion tactics calls for detection systems that analyse traffic patterns beyond traditional signature-based and IP-based methods. Notably, since specific patterns emerge from the malicious network data, our detection mechanisms must leverage this information.

6.3 Quality of Data

This section addresses key challenges and limitations associated with data quality, exploring how these factors influence the integrity and efficacy of our analysis. We will discuss the impact of data capture duration and variability in data collection, which are critical for developing reliable ML models and ensuring accurate anomaly detection.

6.3.1 Period of Recorded Data

The duration of data capture significantly impacts the quality of benign data used in our analysis. While capturing data over 24 hours provides a comprehensive dataset beneficial for robust ML training, it also substantially increases resource demands. Since most benign network traffic comprises short-duration flows, often less than 10 KB and lasting a few hundred milliseconds [47], a 24-hour capture period may not always be necessary.

Several factors influence the decision on capture duration. For endpoint user traffic, capturing data over an entire day or more may be essential to accommodate daily behavioural variations and ensure comprehensive coverage. In controlled environments like corporate network segments with repetitive tasks, such as the one we examine for Recorded Future, 24 hours or shorter may be optimal for capturing a complete cycle of network interactions.

In contrast, the analysis of malicious activities, particularly those involving persistent backdoor connections, may benefit from extended capture periods to fully observe potential data exfiltration and keep-alive connections. The primary goal is to ensure that the capture duration aligns with the full range of observable traffic patterns, thereby improving the accuracy and reliability of anomaly detection models. However, the limitations imposed by Triage's sandbox environment, which restrict our malicious samples to a maximum of 60 minutes of runtime, can truncate longer-lasting behaviours, potentially skewing our analysis.

Disparities in data collection durations across datasets can further influence the variance and comparability of flow duration, as seen when comparing normalised with non-normalised durations in Table 5.4 and Table 5.3. The non-normalised durations would indicate that the benign data had longer duration flows than malicious, on average. This is to be expected since it was captured over a 24 times longer period. After normalising the durations we find that the opposite trend emerges from the data. This shift in trend highlights the need to consider capture durations when examining your data and underscores the need for customised capture strategies based on expected network behaviour.

6.3.2 Variation of Benign Data

A major challenge in developing an optimal detection model is the variability of benign data. For instance, detection frameworks tend to perform better when network traffic patterns are predictable and consistent [1]. Such predictability aids in training ML models by clearly defining "normal" traffic. Focusing on specific network subnets with uniform traffic would produce the best outcome since backdoor communications, which can mimic a range of benign traffic types, inherently complicate detection. For example, the benign data we utilised from Recorded Future originated from two specific network segments designed to perform well-defined tasks. This intentional segmentation within their network infrastructure allowed us to obtain data with well-defined characteristics of normal traffic. Consequently, this coherent data simplified the task of training our model to accurately identify what

constitutes normal traffic patterns.

6.3.3 Variation of Malicious Data

A major hurdle in this project was obtaining high-quality network data from Linux-based backdoor malware. To mitigate the risk of using outdated malware with inactive C2 server connections, we opted for Triage, the sandbox platform discussed in 2.2.2. This platform was chosen because it offers a substantial repository of submitted malware samples, which we could filter to obtain active network communications from backdoor malware. This approach significantly reduced the chances of encountering "stale" malware, which is a common issue when sourcing and running malware samples independently.

Furthermore, dealing with the diverse dependencies that different malware strains require within Linux systems poses a significant challenge in setting up a comprehensive malware analysis environment. Each strain may need specific libraries, kernel versions, or other system configurations to function correctly, which can complicate the process of preparing a unified analysis setup. Given this complexity, our decision to use Triage data was significantly influenced by its extensive dataset of analysed samples that already accounted for these diverse dependencies. We believe that by leveraging such a sandbox environment, we were able to capture a broader spectrum of malicious activities and network traffic patterns.

The PCAP files of malware samples we analysed were collected over varying periods, as reflected in the submission times on Triage. This variability is likely a contributing factor to the distribution of flows we observed, as illustrated in Figure 5.1. The figure shows the final distribution of flow occurrences per malware sample. In the data prepossessing step, the malicious samples were analysed to ensure that no single sample disproportionately influenced the overall data. Some samples were found that had a very large number of flows, such that including them would skew the data and make it inherently biased towards one sample. All such samples were removed. Our objective was to maintain a balance between preserving a sufficient number of malicious samples and ensuring diverse network traffic without bias from dominant samples.

6.3.4 The Need of an ML-Based Detection Model

It could be contended that, given Triage's existing capability to classify samples as malicious, the relevance of an additional detection model may be questioned. However, this assumption overlooks several important considerations. Firstly, the analysis presupposes possession of the actual malware sample file, implying that the file is already suspected to be malicious by its holder. Moreover, our ML-based detection models exclusively analyse network traffic data without performing static or dynamic analysis or delving into binaries. Consequently, it can be regarded as a more lightweight detection model. The model's reliance on network traffic data alone enables it to detect threats in environments where access to file systems is restricted or where malware attempts to evade detection through fileless techniques.

6.4 Flow-Based Versus Packet-Based Data

Our project focused on using the features of flow-based data to identify malicious instances among numerous benign ones. Notable features that we found important during feature selection included duration, the ratio of outgoing to incoming packets, protocols, TCP flags, and the average size of packets in bytes. Both this project and several related works have demonstrated that features from flow-based data are sufficient to classify malicious network traffic effectively. We could effectively identify differences between malicious and benign traffic in the features from NetFlow data as shown in 5.1. Give, the results of the model training and evaluation one could also deduce that the models themselves effectively could use the Netflow features to classify flow-data.

However, there are nuances that packet-based data can reveal, which are lost with flow-based data. For example, time series analysis of when incoming packets arrive and outgoing packets are sent can uncover potentially interesting patterns. This detailed packet-level information is missed when packets are grouped into a single flow representing an overall connection. Additionally, with the granularity of packet data, one could analyse patterns in encrypted payload data, which might provide further insights into malicious behaviour.

Despite these potential benefits, there is one significant downside. In large corporate environments with high volumes of traffic, capturing and storing packets for the duration necessary to conduct thorough analysis poses a substantial challenge. This makes the use of packet-based data difficult to implement in real-life scenarios, where the storage and processing requirements could be prohibitive.

One potential compromise is to capture packets for only long enough to collect the data that is most interesting, focusing on small, essential portions of each packet rather than the entire payload. This approach could provide sufficient data for both time-series analysis and pattern recognition, or other favourable packet-based analysis while still leveraging the benefits of the flow-based approach. By just capturing just the quantitative aspects needed for further analysis, it may be feasible to combine the detailed insights of packet-based data with the efficiency and manageability of flow-based data. However, the feasibility of this would need to be assessed in relation to magnitude of data and the specific data of interest from the packet-captures as well as the computational capabilities of the real-world environments this system would be utilised within.

Moreover, there may be applications of packet-based data related to discovery of potential evasion techniques that adversaries might use to bypass detection. Malicious actors could deliberately design their traffic to mimic benign patterns identified by flow-based data analysis, thereby avoiding detection. Packet-based data, with its finer granularity, could perhaps be used to help detect such sophisticated evasion techniques by analysing subtle variations in packet timing, sequence, and payload content that would not be accessible in flow-based data. Incorporating both flow-based and selective packet-based analysis could provide a more comprehensive defence against these advanced evasion strategies.

6.5 Practical Implications

Benign data obtained from a dedicated network segment of Recorded Future was captured in 24 hours. The network produced **2 980 741** flows and **204.9 MB** after filtering out all the columns and data that were deemed to not be relevant to the model detection in our method. As mentioned in Section 6.1, the best-performing models for SMOTE and without SMOTE were Random Forest and ADABoost. Depicted in Table 6.1, we can compare the practical implications of what each FPR associated with a model means in terms of false alarms that analysts monitoring the detection system would have to confirm or deny to be malicious.

Table 6.1: Amount of false alarms with 2 980 741 flows daily.

ML Model	**FPR**	**# of Flows Flagged**
Random Forest (SMOTE)	0.000185	551
ADABoost (SMOTE)	0.000263	783
ADABoost (NO SMOTE)	0.000029	86
Random Forest (NO SMOTE)	0.000007	20

If we were to perform this calculation for all Netflow data collected daily from within Recorded Futures infrastructure - which amounts up to circa **1.428 GB** and **20 865 187** flows after preprocessing - the FPR of Random Forest (No SMOTE) would generate 146 false alarms.

6.5.1 Vulnerability to Evasion Attacks

The robustness of ML models used for network anomaly detection is a critical concern, particularly in the context of evasion attacks. These attacks are deliberate attempts by adversaries to manipulate input data in ways that cause the ML model to misclassify malicious activities as benign, thereby evading detection. For instance, they might adjust the timing, size, and volume of data packets to fall within the thresholds considered normal by the ML models. This can be particularly effective against models that rely heavily on specific feature thresholds for classification. Given the sophisticated tactics used by modern malware, including sandbox evasion, the deployed models in this thesis are not immune to such threats.

To mitigate these vulnerabilities, future research and model development should focus on enhancing the generalisation and robustness of ML models including:

- Incorporating features that are less susceptible to manipulation, such as statistical anomalies and patterns over time. This can improve model resilience.
- Training models using adversarial samples specifically designed to evade detection. This can help models learn to recognise and mitigate evasion techniques.

- Utilising ensemble learning methods, which combine multiple ML models. This can improve detection rates and reduce the impact of evasion attacks by leveraging the strengths of various models and reducing the likelihood that an attacker can fool all models simultaneously.
- Regularly updating models with fresh data, including new malicious samples and benign traffic. This can help maintain their effectiveness in the face of evolving threats.

6.5.2 Deployment and Cost

This section delves into the duration required for preprocessing raw input data, such as aggregated NetFlow logs, and for the model to generate predictions on the entire dataset. Understanding the cost implications of deploying real-time observations is crucial for assessing the feasibility and scalability of implementing ML-based detection methodologies in network traffic classification.

The time complexity of machine learning models during real-time prediction, after training, can vary depending on factors such as the size of the input data, the complexity of the model, and the hardware on which it's running.

Table 6.2 presents the duration required for preprocessing raw input data, in the form of aggregated NetFlow logs, and for the model to generate predictions on the entire dataset. Given raw NetFlow logs, the preprocessing step is as follows:

1. Extract the relevant NetFlow features. In our case, these are *td, pr, flg, ibyt, obyt, ipkt* and *opkt*.
2. Convert dtypes.
3. Calculate new features. In our case, these are *IbytByIpkt, ObytByOpkt* and *OpktByIpkt*.
4. Replace NaN, inf and -inf with 0.
5. Map *pr* and *flg* fields to integers.
6. Scale the dataset using RobustScalar.

Table 6.2: Time elapsed for prepossessing and predictions for each model.

Model	Total Flows	Elapsed Time (s)	Time Per Flow (s)
ADABoost	2981481	28.010000	0.000009394394
DT	2981481	5.730000	0.000001921828
GNB	2981481	6.160000	0.000002064713
KNN	2981481	301.790000	0.000101220454
RF	2981481	17.420000	0.000005843360
XGBoost	2981481	6.810000	0.000002283812

Considering just the preprocessing step along with the time it takes for each model to make predictions given a dataset consisting of 2981481 flows, it is possible to argue that the classification of NetFlow traffic data into benign or malicious is quite effective. These results are however based on already trained ML models.

In the deployment of ML models within real-time systems, careful consideration must be given to the rate at which NetFlow data is acquired. For instance, the nfcapd tool, designed for the capture of NetFlow data from network devices, typically writes data to disk at intervals of one minute by default. Moreover, by default, nfcapd enacts file rotation every five minutes, thereby generating a new file at five-minute intervals to accommodate the stored NetFlow data. It is noteworthy that these default configurations are adjustable to suit specific requirements. Although the quantity of flows contained within each nfcapd file varies in accordance with the volume of network traffic, it is evident that the utilisation of NetFlow logs in conjunction with a ML-based detection methodology for network traffic classification is, from a time wise standpoint, an efficient approach.

To summarise, analysing flows in real-time might be suitable, as the preprocessing of data and classification of flows is rather time-efficient. Real-time can be defined as, as often that flows are exported from the entity observing the network, and this interval is customisable. Thus, the cost of implementing real time-observations boils down to the magnitude of traffic, as well as how often one will apply the prepossessing and classification steps.

6.5.3 Ethical Aspects

Given the sensitive nature of cybersecurity research, particularly in the context of malware detection, it is imperative to prioritise privacy and data security throughout the process. Respecting user privacy and confidentiality is crucial, especially when dealing with potentially sensitive network traffic data. Additionally, in determining what is to be investigated and developed, it is essential to consider the potential societal impact of the research outcomes. While the focus on enhancing network security is commendable, researchers must also weigh the potential consequences, ensuring that their work does not inadvertently contribute to surveillance, discrimination, or other harmful outcomes.

7

Conclusion

This study emerged due to the insufficient research existing on machine learning-based models for categorising network traffic in Linux-based malware. The study focused solely on the analysis of malicious communication through backdoors.

This research confirms the feasibility of using NetFlow data to detect Linux backdoor communications effectively. By leveraging machine learning methodologies, we achieved a commendable balance of low false positives and satisfactory recall metrics. Ensemble models such as Random Forest and XGBoost showed superior performance, effectively managing the trade-offs between precision and recall. The application of SMOTE generally reduced the false negative rate across most models, enhancing their ability to detect actual anomalies. However, this came at the cost of decreased precision due to the synthetic amplification of the minority class, which increased the number of false positives.

The practical implications of these findings highlight the importance of maintaining a comprehensive and high-quality dataset. The challenges associated with data imbalance and variance in capture durations underscore the need for careful data selection and usage to preserve necessary variation without compromising quality. Future research should focus on expanding the collection of high-quality Linux backdoor samples and refining network data collection methodologies to improve the robustness and reliability of machine learning models in cybersecurity.

Bibliography

[1] Y. Zhang and V. Paxson, "Detecting backdoors," *USENIX Association*, 2000.

[2] B. Lenaerts-Bergmans, *Backdoor attacks*, Jul. 2023. [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/attack-types/backdoor-attack/.

[3] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "Netflow datasets for machine learning-based network intrusion detection systems," in 2021, pp. 117–135. DOI: 10.1007/978-3-030-72802-1_9.

[4] CISCO. "Cisco IOS NetFlow." (2023), [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[5] G. M. W. Al-Saadoon and H. M. Y. Al-Bayatti, "A comparison of trojan virus behavior in linux and windows operating systems," *CoRR*, vol. abs/1105.1234, 2011. arXiv: 1105.1234. [Online]. Available: http://arxiv.org/abs/1105.1234.

[6] H. Nguyen, T. Nguyen, D. Kim, and C. Deokjai, "Network traffic anomalies detection and identification with flow monitoring," Jun. 2008, pp. 1–5, ISBN: 978-1-4244-1979-1. DOI: 10.1109/WOCN.2008.4542524. [Online]. Available: https://ieeexplore.ieee.org/document/4542524?arnumber=4542524.

[7] J. Lindner, *Linux user statisticcs*, May 2024. [Online]. Available: https://worldmetrics.org/linux-user-statistics/.

[8] J. Barath, "Network behavior analysis of selected operating systems," IEEE, Oct. 2019, pp. 1–5, ISBN: 978-80-8040-575-5. DOI: 10.23919/KIT.2019.8883302.

[9] D. H. Hagos, A. Yazidi, O. Kure, and P. E. Engelstad, "A machine-learning-based tool for passive os fingerprinting with tcp variant as a novel feature," *IEEE Internet of Things Journal*, vol. 8, pp. 3534–3553, 5 Mar. 2021, ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3024293.

[10] A. Aksoy, S. Louis, and M. H. Gunes, "Operating system fingerprinting via automated network traffic analysis," IEEE, Jul. 2017, pp. 2502–2509, ISBN: 978-1-5090-4601-0. DOI: 10.1109/CEC.2017.7969609.

[11] D. Delaney, *Comparing packet and flow capture*, en, Oct. 2012. [Online]. Available: https://www.computerworld.com/article/2473229/comparing-packet-and-flow-capture.html.

[12] IBM, *Netflow versions*, 2022. [Online]. Available: https://www.ibm.com/docs/en/npi/1.3.1?topic=reference-netflow-versions.

[13] I. Cisco Systems, *Netflow data export formats user guide*, 2007. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/5-0-3/user/guide/format.html#wp1006108.

[14] P. Haag, *Nfdump*, https://github.com/phaag/nfdump, Accessed: May 20, 2024, 2024-02-17.

[15] T. T. Inc., *Triage: Automated Threat Intelligence Platform*, https://tria.ge/, Accessed: May 20, 2024, 2024.

[16] V. Alvarez, *Yara*. [Online]. Available: https://virustotal.github.io/yara/.

[17] Mitre, *Mitre*. [Online]. Available: https://attack.mitre.org/.

[18] Mitre, *Command and control*. [Online]. Available: https://attack.mitre.org/tactics/TA0011/.

[19] Mitre, *Command and control*. [Online]. Available: https://attack.mitre.org/tactics/TA0010/.

[20] R. Grimmick, *What is c2? command and control infrastructure explained*. [Online]. Available: https://www.varonis.com/blog/what-is-c2.

[21] J. Barnard and C. Stryker. "What is anomaly detection?" Published: December 12, 2023. Accessed: May 20, 2024. (2023).

[22] C. P. Bathula, *Machine Learning Concept 53: XGBoosting & Adaboosting*, https://medium.com/@chandu.bathula16/machine-learning-concept-53-xgboosting-adaboosting-663cd8c920e2, Accessed: May 20, 2024, Mar. 2023.

[23] Mahesh, *Exploring Decision Trees, Random Forest, Logistic Regression, KNN, Linear Regression, SVM, RNN, LSTM, and LightGBM for Effective Data Analysis*, https://medium.com/@maheshhkanagavell/exploring-decision-trees-random-forest-logistic-regression-knn-linear-regression-svm-rnn-bf52fa94a066, Accessed: May 20, 2024, Jun. 2023.

[24] "F statistic / f value: Simple definition and interpretation." Accessed: May 20, 2024, Statistics How To. (Unknown), [Online]. Available: https://www.statisticshowto.com/probability-and-statistics/f-statistic-value-test/.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011, Accessed: May 20, 2024.

[26] S. Sahazada, *Correlation-based feature selection in a data science project*, Medium, "Accessed: May 20, 2024.", Mar. 2023. [Online]. Available: https://medium.com/@sariq16/correlation-based-feature-selection-in-a-data-science-project-3ca08d2af5c6.

[27] S. Satpathy. "SMOTE for Imbalanced Classification with Python." Accessed: May 20, 2024. (2023), [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/.

[28] T. Srivastava. "12 Important Model Evaluation Metrics for Machine Learning Everyone Should Know." Accessed: May 20, 2024. (2024), [Online]. Available: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/.

[29] K. Muralidhar. "What is Stratified Cross-Validation in Machine Learning?" Accessed: May 20, 2024. (2021), [Online]. Available: https://towardsdatascience.com/what-is-stratified-cross-validation-in-machine-learning-8844f3e7ae8e.

[30] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, vol. 86, pp. 147–167, 2019, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.06.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740481930118X.

[31] H. Ahmetoglu and R. Das, "A comprehensive review on detection of cyber-attacks: Data sets, methods, challenges, and future research directions," *Internet of Things*, vol. 20, p. 100 615, 2022, ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2022.100615. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S254266052200097X.

[32] T. Thomas, A. P. Vijayaraghavan, and S. Emmanuel, *Machine Learning Approaches in Cyber Security Analytics*, 1st ed. Springer Singapore, 2020, p. 209, ISBN: 978-981-15-1706-8. DOI: 10.1007/978-981-15-1706-8. [Online]. Available: https://link.springer.com/book/10.1007/978-981-15-1706-8.

[33] S. Das, S. S. Mullick, and I. Zelinka, "On supervised class-imbalanced learning: An updated perspective and some key challenges," *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 6, pp. 973–993, 2022. DOI: 10.1109/TAI.2022.3160658. [Online]. Available: https://ieeexplore.ieee.org/document/9738474.

[34] A. B. Nassif, M. A. Talib, Q. Nasir, and F. M. Dakalbab, "Machine learning for anomaly detection: A systematic review," *IEEE Access*, vol. 9, pp. 78 658–78 700, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235308115.

[35] R. Grimmick, *Packet capture: What is it and what you need to know*. [Online]. Available: https://www.varonis.com/blog/packet-capture.

[36] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," IEEE, Nov. 2015, pp. 1–6, ISBN: 978-1-4673-7007-3. DOI: 10.1109/MilCIS.2015.7348942.

[37] C. I. for Cybersecurity, *Cse-cic-ids2018 on aws*, 2018. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html.

[38] M. Rodríguez, A. Alesanco, L. Mehavilla, and J. García, "Evaluation of machine learning techniques for traffic flow-based intrusion detection," *Sensors*, vol. 22, p. 9326, Nov. 2022. DOI: 10.3390/s22239326.

[39] D. Jiang and K. Omote, "An approach to detect remote access trojan in the early stage of communication," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015, pp. 706–713. DOI: 10.1109/AINA.2015.257.

[40] A. Vega, I. Crespo-Martínez, Á. Guerrero-Higueras, C. Álvarez-Aparicio, V. Matellán, and C. Fernández, "Malicious traffic detection on sampled network flow data with novelty-detection-based models," *Scientific Reports*, vol. 13, Sep. 2023. DOI: 10.1038/s41598-023-42618-9.

[41] K. S. Yin and M. A. Khine, "Optimal remote access trojans detection based on network behavior," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, p. 2177, 3 Jun. 2019, ISSN: 2088-8708. DOI: 10.11591/ijece.v9i3.pp2177-2184.

[42] OpenAI, *Chatgpt: Language model by openai*, https://www.openai.com/chatgpt, Accessed: 2024-06-02, 2023.

[43] R. Future, *Ip address cards*. [Online]. Available: https://www.recordedfuture.com/support/ip-address-cards.

[44] Mitre, *Dynamic resolution*. [Online]. Available: https://attack.mitre.org/techniques/T1568/.

[45] Mitre, *Virtualization and sandbox evasion*. [Online]. Available: https://attack.mitre.org/techniques/T1497/.

[46] Mitre, *Execution flow hijacking*. [Online]. Available: https://attack.mitre.org/techniques/T1574/.

[47] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," ACM, Nov. 2010, pp. 267–280, ISBN: 9781450304832. DOI: 10.1145/1879141.1879175.