Lancaster University

Enhancing Anomaly Detection Techniques for Emerging Threats

Ryan Michael Mills

This dissertation is submitted for the degree of Doctor of Philosophy

Abstract

Despite the Internet being an apex of human achievement for many years, criminal behaviour and malicious activity are continuing to propagate at an alarming rate. This juxtaposition can be loosely attributed to the myriad of vulnerabilities identified in existing software. Cyber criminals leverage these innovative infection and exploitation techniques to author pervasive malware and propagate devastating attacks. These malicious actors are motivated by the financial or political gain achieved upon successful infiltration into computer systems as the resources held within are often very valuable in nature.

With the widespread developments in the Internet of Things (IoT), 5G, and Starlink satellites, unserved areas of the world will experience a pervasive expansion of connected devices to the Internet. Consequently, a barrage of potential new attack vectors and victims are unfolding which requires constant monitoring in order to manage this ever growing problem. Conventional rule-based intrusion detection mechanisms used by network management solutions rely on pre-defined attack signatures and hence are unable to identify new attacks. In parallel, anomaly detection solutions tend to suffer from high false positive rates due to the limited statistical validation of ground truth data, which is used for profiling normal network behaviour.

When considering the explosive threat landscape and the expanse of connected devices, current security solutions also face challenges relating to the scale at which attacks need to be monitored and detected. However, recent innovations in Big Data processing have revealed a promising avenue in which scale is addressed through cluster computing and parallel processing.

This thesis advances beyond current solutions and leverages the coupling of anomaly detection and Cyber Threat Intelligence (CTI) with parallel processing for the profiling and detection of emerging cyber attacks. This is demonstrated through the design, implementation, and evaluation of *Citrus*: a novel intrusion detection framework which is adept at tackling emerging threats through the collection and labelling of live attack data by utilising diverse Internet vantage points in order to detect and classify malicious behaviour using graph-based metrics, as well as a range of Machine Learning (ML) algorithms.

This research provides innovative contributions to the cyber security field, including the public release of an open flow-based intrusion detection data set. This data set encompasses emerging attack patterns and is supported by a robust ground truth. Furthermore, Citrus advances the current state of the art through a novel ground truth development method. Citrus also enables both near real-time and offline detection of emerging cyber attacks under optimal computational costs. These properties demonstrate that it is a viable and practical solution for next generation network defence and resilience strategies.

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university.

Ryan Michael Mills
September, 2021

Acknowledgements

I would like to begin by expressing my sincere gratitude to my PhD supervisors, in particular Dr. Matthew Broadbent, who have been pillars of support during my research. I would also like to thank Dr. Angelos Marnerides for his considerable insight into the world of cyber security and anomaly detection.

Outside of the sphere of research, I am eternally thankful to my family. In particular my grandfather, David Mills, who inspires me to be successful.

Contents

List of Figures

List of Tables

List of Acronyms

ACCS	Australian Centre for Cyber Security
AD	Anomaly Detection
AES	Advanced Encryption Standard
AP	Access Point
API	Application Programming Interface
APT	Advanced Persistent Threat
AS	Autonomous System
CC	Command and Control
CADHo	Collection and Analysis of Data from Honeypots
CDF	Cumulative Distribution Function
CSV	Comma-separated values
CTI	Cyber Threat Intelligence
CTIS	Cyber Threat Intelligence Service
DDoS	Distributed Denial of Service
DGA	Domain Generation Algorithm
DNN	Deep Neural Network
DoS	Denial of Service
EMBER	Endgame Malware BEnchmark for Research
ENISA	European Union Agency For Cybersecurity
FN	False Negative

FP	False Positive
FTP	File Transfer Protocol
HDFS	Hadoop File System
HIDS	Host-based Intrusion Detection System
HTTPS	Hypertext Transfer Protocol Secure
ICS	Industrial Control System
IDS	Intrusion Detection System
IP	Internet Protocol
IRC	Internet Relay Chat
IV	Initialisation Vector
IoC	Indicator of Compromise
IoT	Internet of Things
JSON	JavaScript Object Notation
LAN	Local Area Network
ML	Machine Learning
MLLib	Machine Learning Library
MiTM	Man in The Middle
NAT	Network Address Translation
NHS	National Health Service
NIDS	Network-based Intrusion Detection System
NSA	National Security Agency
OSINT	Open-source Intelligence
P2P	Peer to Peer

PCAP	Packet Capture
PE	Portable Executable
PII	Personally Identifiable Information
PoC	Proof of Concept
RBF	Radial Basis Function
RCE	Remote Code Execution
RDD	Resilient Distributed Dataset
RPCL	Rival Penalized Competitive Learning
SCP	Secure Copy
SDN	Software-defined Networking
SQL	Structured Query Language
STIX	Structured Threat Information eXpression
SVM	Support Vector Machine
TAXII	Trusted Automated eXchange of Indicator Information
TCP	Transmission Control Protocol
TTL	Time To Live
U2R	User to Root
UDP	User Datagram Protocol
UPnP	Universal Plug n Play
VMI	Virtual Machine Introspection
VNC	Virtual Network Computing
WAN	Wide Area Network
XSS	Cross Site Scripting

Publications

The work presented in this thesis has been published in the following journal and conference:

R. Mills, A. K. Marnerides, M. Broadbent and N. Race. Practical Intrusion Detection of Emerging Threats. *IEEE Transactions on Network and Service Management*. 2021. DOI: 10.1109/TNSM.2021.3091517.

R. Mills, N. Race and M. Broadbent. Citrus: Orchestrating Security Mechanisms via Adversarial Deception. *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium.* 2020. DOI: 10.1109/NOMS47738.2020.9110443.

Chapter 1

Introduction

The advancement in technology has paved the way for the pervasive integration of computers into the lives of many. The respective data and resources they contain makes each a potential target for attackers. Disruption to these systems can incur financial losses for operators, and wider consequences for users who are unable to access resources.

The cost of damages caused by malicious activity over the Internet is rapidly growing, and by 2021 the estimated cost of attacks orchestrated throughout computer networks is assessed to be around $6 trillion per annum [1]. Furthermore, with the substantial growth in active malware campaigns, this overall trend does not seem to be stopping. Network monitoring solutions are typically used to tackle these attacks. To defend against this growth in unique distributed infections, the method in which attacks are identified and prevented have recently received significant overhaul.

As well as the number of attacks increasing, the explosive growth of connected devices and the corresponding traffic generated in modern networks also create challenges for traditional network monitoring solutions. Networks which compose the Internet have recently been reported to be generating data at the scale of zettabytes. This is further exacerbated by the widespread expansion of the Internet of Things (IoT), with estimates forecasting the number of heterogeneous devices connected to Internet Protocol (IP) networks to be three times more than the global population by 2023 [2]. Hence, modern network traffic satisfies the requirements of big data, as it contains properties of vast volume, variety, and velocity [3].

An emerging approach in the prevention of cyber crime is the use of machine learning algorithms to perform intrusion detection. This approach to anomaly detection attempts to define normal behaviour through the training of models, and any substantial deviation from that baseline identifies a potential infiltration attempt. However, there often occurs a number of false positives due to challenges related to the profiling of normal behaviour within training data [4].

In parallel, there has also been substantial developments in the evaluation of attacker behaviour through the use of Cyber Threat Intelligence (CTI) services. The knowledge gained by the contextualisation of observed data enables a fuller understanding of the attack and actor as a whole, leading to a more accurate conclusion of malicious intent. Leveraging this understanding of malicious behaviour grants insight into the true nature behind suspicious activity, paving the way for the establishment of a ground truth.

The ground truth provides a method to separate benign from attack traffic, thus revealing the true nature of data captured within networks. This establishment of a ground truth aids machine learning algorithms to more accurately identify patterns of normal and malicious behaviour, reducing the number of False Positives (FPs) and False Negatives (FNs) obtained under decision making scenarios.

1.1 Contemporary Intrusion Detection

One of the primary defense mechanisms of choice within modern enterprise networks is typically an Intrusion Detection System (IDS). An IDS performs classification of events which transpire within a network into either benign or malicious in an attempt to thwart intrusion attempts. These events can be network based, such as packets or flows traversing a switch, or host based, such as process utilisation and system logs. This classification can then be leveraged to inform an administrator of malicious behaviour or perform remediation through an automated response.

The decision making process can be categorized as either *misuse* or *anomaly* detection. Misuse detection uses pre-defined attack patterns using signatures of known malware. As a result, novel attacks exploiting unknown vulnerabilities bypass this approach as there exists no corresponding signature. Furthermore, signatures are often too specific and minute code alteration of malware also circumvent the decision making process. Misuse detection relies upon a database of known attack signatures, which is necessarily large and requires constant updates to keep abreast of developing threats. AD (Anomaly Detection) leverages a description of normal behaviour, which is learned through observations in training data, and any future observation which deviates from the normal baseline is labeled as malicious. Nonetheless, the task of profiling normal behaviour is highly challenging due to its dependency on the establishment of ground truth data, which in many cases are not validated either statistically or even empirically. Hence, anomaly detection solutions can also be problematic especially when a large volume of information is processed and statistical normality is not thoroughly assessed.

The unprecedented growth in the number of Internet users, which is estimated to be 66% of the overall population in 2023 [2], has caused an exponential increase in the generation and transmission of network traffic. This has naturally led to an era of big data within modern networks, which poses numerous challenges in the sphere of network security [5]. Big data is typically defined in terms of the three Vs: volume, variety, and velocity. As originally specified in [6], volume refers to the sheer quantity of data. Velocity refers to the speed at which data is required to be processed, which certainly causes challenges when data is transmitted through networks at a rate greater than the capacity of traditional monitoring solutions [7]. Variety refers to the assortment of heterogeneous data. Within modern networks, this is typically manifested by IoT devices, which often leverage diverse communication protocols.

The requirements distilled by the nature of the emerging cyber threat landscape demonstrate that attacks are now conducted on a large scale, thus IDS-based solutions are required to maintain vast quantities of either signatures or training data used for decision making. Furthermore, the decision making procedure is additionally hindered by the copious quantity of traffic in modern networks which needs to be processed. It is essential to immediately process this data within the network for the rapid identification of anomalies. However, traditional approaches are incapable of handling big data [7, 5, 8]. For example, 85% of cyber attacks are detected weeks after they occur, with an average detection time of 206 days [9].

Therefore, a high throughput processing framework is required to adequately analyse and classify this data in a reasonable time frame. Big data frameworks such as Hadoop [10], Google BigTable [11], Elasticsearch [12] are inherently scalable by design and thus able to deal with extremely large data sets. However, they lack high throughput processing capabilites. Recently, Spark [13] has emerged as a prominent distributed computing framework, which has evidenced the ability to handle the large amount of data required for contemporary intrusion detection [14, 5].

Spark enables the parallel processing of vast data sets through the segmentation of data amongst nodes within a cluster. Spark also offers streaming capabilities suitable for online anomaly detection using telemetry gathered from internal hosts. This technology critically enables data to persist in memory, leading to substantial efficiency improvements when compared to conventional cluster computation frameworks such as MapReduce [15]. This improvement is further exacerbated when considering iterative algorithms, such as those used in machine learning. Providing anomaly detection algorithms the ability to scale with the influx of emerging threats is crucial to overcome the challenges brought about by the rapid growth of connected devices, innovative infection techniques, and the large variety and volume of data within modern networks [7].

1.2 Cyber Threat Intelligence

CTI refers to the behaviour and information derived from the observation of threat sources. Current research into this sphere typically attempts to extract Indicators of Compromise (IoCs) to gain an understanding of attack properties so attacks of the same type can be prevented. An invaluable threat intelligence source is a honeypot. Honeypots are systems under observation, which contain components that masquerade as legitimate enterprise infrastructure in order to catch unsuspecting adversaries leveraging previously unobserved exploits, attack tactics and patterns used for infiltration [16]. Deploying a honeypot on the Internet allows one to gain a point of observation, which, when leveraged properly, grants unrestricted access to activity initiated by legitimate benign services (e.g. Shodan [17]) as well as malicious sources such as botnets. This insight into the emerging threat landscape enables IDS's the ability to store relevant knowledge about bleeding edge compromise methodology. Furthermore, the decision engine of the IDS is bolstered with recent data enabling it to make a better informed choice on whether or not an action is malicious.

When using supervised machine learning algorithms to determine abnormality in systems telemetry, it is essential to train the models on data, which includes benign behaviour. However, the training data must also include relevant attacks, which are likely to be encountered and need to be identified and remediated. There exists challenges with contemporary intrusion detection data sets such as the fact that they contain dated and non-relevant attacks [18], and manually injected attacks, which do not accurately reflect the current threat landscape [19]. Training data derived from honeypot telemetry is a solution to these challenges posed by the emerging threat landscape [20], as the data contained within is representative of a partial view of the malicious actions currently propagating throughout the Internet.

Furthermore, one of the most crucial features in supervised machine learning algorithms is the ground truth represented as target labels. These labels are used to inform algorithms about the true nature of the data. Current data sets often do not either include this feature [21], or engineer it in a non-standard and often problematic manner [22, 23]. This often forces the use of unsupervised statistical methods, or in the case of ineffective labelling techniques, the trained supervised models will often have a distorted view of normal behaviour, and the detection results will be impacted accordingly. Honeypots not only attract malicious traffic, but also legitimate traffic such as scanners attempting to profile the Internet topology [24]. Thus, labelling all traffic associated with honeypots as malicious could negatively affect the decision making process of AD algorithms when used as training data.

Current solutions which integrate CTI services with IDSs have not yet explored the possibility of the development of a ground truth to provide accurate labels for telemetry captured by heterogeneous honeypot deployments. Services such as Greynoise [25] and Censys [26] offer an open API (Application Programming Interface) to gather contextual information about a suspicious host. They typically achieve this by deploying monitoring nodes within data centres to identify web crawlers, spam, botnet, and other malicious activity. This contextual information offers corroborating evidence in the identification of malicious actors as they confirm whether they are displaying malicious behaviour in other areas of the Internet. Naturally, this additional context heralds innovation for a novel ground truth development method for use in the labelling of data sets. Fundamentally, this is intended to aid anomaly detection algorithms in the accurate identification of malicious behaviour.

1.3 Motivation

The emergence of novel threats leveraging sophisticated exploits has caused devastating damage to computer networks. In recent events, this has been realised through extensive botnets performing mass exploitation to steal sensitive documents and service disruption caused by large scale Distributed Denial of Service (DDoS) attacks utilising innovative methods of infection. This issue coupled with the explosive growth of the Internet, leading to increased attack vectors, poses challenges to current security solutions.

Through the utilisation of emerging technologies and innovative ground truth development techniques, this thesis aims to present a solution for intrusion detection in the modern Internet age. In detail, this solution incorporates the use of big data frameworks to address scalability concerns inherent within modern networks. Moreover, automatic attack telemetry collection and labelling is undertaken, through the composition of honeypots and correlation with CTI services, in order to establish a ground truth, which aims to improve the accuracy of detection mechanisms. Furthermore, the latter approach additionally provides a constant feed of emerging threat data, which could be used to iteratively train machine learning classifiers, enabling an evolving understanding of malicious behaviour, and has been suggested to solve the current issues faced by public IDS data sets [23].

1.4 Research Questions

Motivated by such events in recent times, this thesis aims to answer several open questions within research. In detail, these include:

- **Research Question 1:** Can a continuously updated intrusion detection data set be authored through the deployment of honeypots?

IDSs require up-to-date profiles of malicious behaviour in order to defend against the most cutting edge attack vectors. Most intrusion detection data sets within literature are *not* updated to reflect these emerging attack patterns. As a result, IDSs which leverage these data sets will often not be able to detect attacks that are currently propagating. In order to provide contributions to the security community in the form of updated attack data, this thesis explores the use of honeypots to continuously capture emerging attack patterns. To the best of our knowledge, the use of honeypots has not previously been used in literature to create an updated intrusion detection data set.

- **Research Question 2:** Can CTI services be leveraged to provide accurate labels for telemetry, which contains emerging attack patterns and benign traffic?

Through the composition of honeypots within public address space, emerging attack patterns which are orchestrated over the Internet can be captured. In addition to attacks, honeypots also capture traffic that can be classed as benign. To distinguish between these varying classes of traffic, a ground truth must be developed. The clear separation of this traffic can be used to accurately inform supervised Machine Learning (ML) approaches of both normal and malicious behaviour, leading to more accurate decision making processes. Recent developments within the area of CTI provide evidence of attacks, which have been observed in various parts of the Internet. To the best of our knowledge, CTI has not previously been used to provide accurate target labels for telemetry with the intention to aid anomaly detection approaches to intrusion detection. Due to the lack of research within this area, it is essential to examine whether such an approach is feasible and appropriate.

- **Research Question 3:** Can innovations within Big Data technologies help deal with the scale at which traffic in modern networks is required to be processed for the intrusion detection of emerging threats?

As previously discussed, the amount of traffic which traverses modern networks has grown substantially. In order to identify malicious behaviour within the network, this traffic must be evaluated. Traditional approaches have typically struggled to deal with this influx of data related to the growing number of connected devices. There have recently been innovations within Big Data technologies, which claim to handle extremely large amounts of data with low processing overhead. This thesis intends to explore the benefits provided by these emerging technologies. Specifically, this thesis will evaluate Big Data technologies in the context of intrusion detection, and will assess whether near real-time detection of attacks encompassed within large amounts of data can be achieved through integration with these technologies.

1.5 Thesis Aims and Contributions

This thesis aims to investigate modern intrusion detection. This is achieved through design, implementation and evaluation, with the main aims and contributions of this thesis summarised below:

1. **Attack data availability**: In order to evaluate Citrus, the AD component must be trained on a robust data set incorporating emerging attacks, and normal behaviour. Due to the challenges identified with current data sets during literature review, Citrus orchestrates the collection and processing of emerging threat data through the composition of honeypots scattered throughout Lancaster University's public address space. Citrus then produces an open and reusable data set suitable not only for the evaluation of itself, but also for the evaluation of next generation intrusion detection techniques. This contribution is created to address Research Question 1.

2. **Practical integration of CTI for active network defense**: Supervised ML algorithms require labelled training data to perform classification. In order to provide contributions within this domain, Citrus develops a ground truth for network telemetry data through a novel integration with CTI services. The labelled data created through this process can be used to evaluate next-generation attack detection frameworks. This contribution aims to address Research Question 2.

3. **Near real-time anomaly detection**: Citrus is also designed with tremendous scalability in mind. To provide effective attack detection capabilities in large-scale networks, Citrus is integrated with big data technologies. Citrus is also released in an open-source format to promote adoption by both research and industry. This contribution aims to address Research Question 3.

1.6 Thesis Structure

This thesis is structured into six distinct chapters, the remaining are detailed in the following:

- **Background and Related Work (Chapter 2)**: This chapter initially provides an introduction to the emerging threat landscape by exploring current propagation, exploitation, and communication methodology. This chapter then examines the use of CTI, with a focus on the use of honeypots as a cyber defense strategy. This chapter also provides a detailed description of contemporary intrusion detection methods, including the various different data input types, and detection engine methodology. Finally, the chapter concludes with a discussion of current challenges within this area of research. These challenges are identified with the goal of motivating several design requirements.

- **Design (Chapter 3)**: This chapter discusses the design requirements for effective and efficient intrusion detection derived from the aforementioned related work. This chapter then goes on to showcase the design of a novel intrusion detection framework, which aims to satisfy the challenges posed by the emerging threat landscape via the integration with developing technologies and CTI services.

- **Implementation (Chapter 4)**: This chapter highlights the technical implementation of the previously mentioned design of a contemporary intrusion detection framework. It details the data collection and processing pipeline, alongside a comprehensive description of how the data is classified using ML algorithms. In addition, this chapter includes a presentation of how Citrus is instrumented to establish a ground truth to aid its detection approach.

- **Evaluation (Chapter 5)**: This chapter undertakes rigorous investigation into the efficiency and effectiveness of the detection approach taken, clearly detailing the benefits to scalability and ability to accurately detect emerging threats. It achieves this by initially evaluating the efficiency of the data processing pipeline, and then through the examination of its detection capabilities an evaluation of its ability to detect various attacks is conducted. Furthermore, the ground truth undergoes validation to ensure the correctness of the labels applied to the telemetry used in the training of ML models for detection purposes. This series of evaluations ultimately explore whether the design requirements of Citrus have been met.

- **Conclusion (Chapter 6)**: This chapter summarises the contributions of this thesis, and goes on to document the potential areas in which this work can be extended. Critically, this chapter explores whether the research questions outlined in this chapter have been successfully answered.

Chapter 2

Background and Related Work

The overarching research questions outlined in the previous chapter motivate several aspects of this thesis. In order to answer these questions, investigations must initially be made within several areas of research to identify similar existing approaches. The findings related to this investigation are documented within the following chapter of this thesis. Based upon these, the design, implementation, and evaluation of a prototype is used to explore whether these questions can be answered with a positive outcome.

To begin, a critical examination of the evolving threat landscape and the way in which monitoring and security mechanisms are employed in order to react to these threats is conducted. Section 2.1 begins to present various attack methods, malware families and types, detailing the methodology used to gain a foothold into infrastructure and perform malicious actions at scale over the Internet.

One way in which emerging attacks are able to be monitored and analysed is through the utilisation of CTI. Section 2.2 details various CTI providers, namely honeypots, in recent literature. This includes the ability to collect attack telemetry, malware and generate signatures of compromise.

Through the monitoring and analysis of these threats, efforts can be made to mitigate the problem. Section 2.3 highlights the use of IDS in contemporary literature, examining the different methods used to detect malicious actions. This section additionally investigates available intrusion detection data sets and their deficiencies.

2.1 Modern Threat Landscape

Networked computer systems are increasingly susceptible to abuse due to the growing propagation of exploitation and disruption campaigns throughout the Internet. Motivated by economic and geopolitical gain, these myriad attacks encompass a large range of techniques and are very diverse in nature. Ranging from IoT malware initiating large scale DDoS attacks, to targeted and stealthy intrusions into critical infrastructure leveraging zero-day exploits as part of a sophisticated attack chain, the emerging threat landscape remains a challenging problem to solve.

Within the modern threat landscape exists a variety of evolving and unfolding attack vectors. These type of threats, which contain novel or updated infection properties are known as *emerging threats*. Typically, threats such as these emerge after recent developments within the security community, such as the public disclosure of a new software vulnerability.

2.1.1 Malware

Attack campaigns typically utilise malware in order to automate stages in an attack chain, such as lateral movement and privilege escalation. Malware are typically classified into different families, each with a specific purpose and technique. According to a recent report issued by FireEye, 41% of all observed malware families in 2020 were previously unknown [27]. This emergence of novel malware depicts a sobering insight into the threat landscape. New authors continue to be motivated by lucrative returns and begin development, while current authors are prepared to fully reimplement their malware in an attempt to evade detection and gain infections.

The MITRE ATT&CK knowledge base documents over 240 distinct tactics and techniques used in adversary behavior and taxonomy for adversarial actions across their lifecycle [28]. Since its inception in 2013, this knowledge base has grown to incorporate and reflect the evolving attack vectors discovered in the wild. Malware authors are continuously innovating, employing increasingly complex methods including exploitation, privilege escalation, lateral movement, and data exfiltration in an attempt evade detection and complete their objective.

One of the most effective methods employed in malware is the use of undisclosed vulnerabilities, more commonly known as zero-days, which are capable of exploiting fully patched software. In an unprecedented attack campaign, Stuxnet malware leveraged four unique zero-day vulnerabilities targeting systems operating Windows within an Iranian nuclear power plant [29]. These powerful exploits remain illusive in nature with bug bounties allowing security researchers to benefit economically for their efforts. Despite this, individuals may decide to choose a different approach: weaponising the exploit and incorporating into bespoke malware, or sale of said exploit on the black market, typically returning a larger payout when compared to bug bountys. While initially causing large scale compromise, as evidenced in the WannaCry outbreak, which weaponised the leaked and highly severe EternalBlue exploit [30], zero-days continue to infect devices years after they are disclosed due to systems remaining unpatched [27] [31]. As a result, these exploits are commonly instrumented within emerging malware, despite the vulnerability being disclosed years prior. Being so effective and readily available through open source weaponised code, there has been an emergence of Exploit Kits. Exploit Kits provide a sophisticated delivery method in the distribution of malware, typically targeted at Windows machines with *unpatched* commonly installed software such as Internet Explorer and Java. Exploit Kits are comprised of a number of different exploits. Through the analysis of User Agent strings embedded in HTTP headers, they filter viable candidates and target vulnerable software with a relevant exploit. Next, they attempt to exploit these vulnerabilities, and if successful, the final payload is revealed and executed upon the target machine. Often, Exploit Kits are provided as a PaaS offering. This enables malware authors to focus on functionality, rather than the infection vector.

Spelevo [32] Exploit Kit has been observed propagating through targeted advertising campaigns, injecting malicious iframes into visitors of legitimate websites. This campaign commonly makes use of CVE-2018-15982 [33], an Adobe Flash vulnerability enabling arbitrary remote code execution. In parallel, the Rig [34] Exploit Kit has targeted users of adult websites through an advertising campaign named HookAds. This campaign funnels traffic to decoy adult websites where CVE-2018-8174 [35] is taken advantage of. This vulnerability targets Internet Explorer and exploits the VBScript engine to gain Remote Code Execution (RCE). A report by MalwareBytes disclosed this Exploit Kit has stopped using Flash vulnerabilities, possibly signalling its limited utility due to Flash reaching its end of life at the end of 2020 [36].

Upon successful infiltration of a system, malware typically communicates with Command and Control (C&C) infrastructure to receive commands, or in the case of ransomware, a unique public key is transferred in order to encrypt sensitive documents. Traditionally, botnets have utilised the IRC (Internet Relay Chat) protocol as a C&C medium, with bots instrumented to connect to a predefined server and specified channel. They lay dormant until receiving a command from the bot master, in which case each bot processes the task at once. This process is now rarely observed in the wild, with HTTP now commanding a dominant share of all C&C traffic, typically due to firewalls permitting this protocol access to the Internet, masquerading as legitimate web traffic [37]. This method leverages HTTP requests, with the malware initiating communication with the C&C server, providing system information within the request body. Based on the request the C&C server receives, a tailored response and corresponding commands are sent to the victim.

However, both of these methods suffer from the same problem: a single point of failure. If the server in each case is taken offline through abuse reports, the malware cannot receive commands and effectively becomes benign. Recent innovations in C&C architecture and methodology have been discovered which attempt to address this problem. Through distributed placement of C&C servers, proxy redirection, Peer-to-Peer (P2P) networking, DNS fast-flux, and Domain Generation Algorithms (DGA), malware authors continue to attempt to elude discovery and counter measures.

DGA provides a method in which hostnames corresponding to C&C servers are rotated periodically. In this instance, malware instrumented to communicate with C&C servers first generate the domain through a function using the current time and date as an argument. If the domain generated successfully resolves, the malware designate it as the authoritative command server until the next round of generation. After each iteration, the previous domains are discarded and never used again, ensuring law enforcement cannot perform reactive take downs and cause disruption to functionality. One method used by researchers to monitor this activity is through the analysis of malware. Researchers at University of California, Santa Barbara employed this method to inspect the generation function of emerging TorPig variants [38]. Through identifying the mechanisms used to generate the domain on a given date, they were able to register future domains before the bot masters, effectively taking control of the botnet for 10 days. During the length of control, the researchers were able to intercept communications from bots, and were able to retrieve millions of account details obtained through Torpigs form capture and password manager attacks.

DGA is often used in conjunction with DNS Fast-Flux to provide an additional layer of protection against counter measures. The Fast-Flux concept rapidly changes the IP addresses corresponding to a given domain name. Through the constant update of DNS A records, IP addresses of other infected hosts are inserted with a low TTL (Time To Live) value, cycling them every few minutes. When the malware wishes to communicate with the C&C server, the IP of the given domain is resolved through recursive DNS, with the authoritative server returning an IP address of another infected host. The malware will then connect to the identified remote host, proxying the request to the C&C server.

2.1.1.1 Botnet

Botnets remain a prominent threat in recent times, comprising a vast number of malicious activity currently occurring in computer networks. These threats are capable of infecting large numbers of hosts through powerful exploitation techniques and persistent malware, making them also part of the growing botnet. Botmasters are individuals who control botnets from behind the scenes, orchestrating the infected bots' actions through administration of the C&C servers, providing them a layer of anonymity. Through this process, individual bots or the entire army can be ordered to complete tasks such as updating malware binaries, or performing distributed attacks such as DDoS. Updating of malware binaries is performed to enable additional functionality of the botnet, with the most popular in recent times including click fraud, phishing, spam campaigns, and cryptocurreny mining [39].

Due to the nature of botnets, they are extremely lucrative and are capable of earning large amounts of profit through diverse sources. This is often achieved through financial fraud leveraging form capture, keylogging, or password manager exploit, where the credit card information and banking account credentials are captured and sold for profit or used to extract funds from victims' accounts. This is not always the case, and botnets are capable of earning money by simply controlling large numbers of hosts. As evidenced by the Mariposa botnet, individuals are willing to rent these botnets to coordinate attacks of their own, effectively skipping the large investment needed to gain these infections [40].

Coordinated by C&C infrastructure and malware present on bots, botnets such as Mariposa initiate scanning campaigns in order to identify vulnerable systems via the discovery of open ports. Port scans of this nature can be achieved through UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) methods, with each botnet choosing one suitable for services they are actively trying to exploit.

Utilising TCP as the scanning protocol is a popular approach in botnets, with most modern services operating with this protocol, and it generally being faster than UDP methods. Stealth TCP scans are initiated by sending a prospective host a SYN packet directed to the target port, if a SYN/ACK response is returned, it is determined the port is open, and if a RST packet is received, the port is closed. This technique is relatively unobtrusive as the full TCP connection is not made, limiting the noise involved in the process.

In some cases, botnets also utilise UDP in scanning mechanisms, however this protocol has limitations over TCP implementations, namely open and filtered ports rarely respond, incurring heavy delays and possible retransmission if the packet was thought to be lost. Furthermore, this problem is exemplified when combined with the fact that closed ports usually send back ICMP Port Unreachable errors, which are also often rate limited by many systems including the Linux kernel. In order to work around this issue, bots will typically scan the same port of many different hosts simultaneously, therefore, circumventing rate limit limitations. This is known as *horizontal* scanning, and many popular botnets are currently employing this technique to scan vast numbers of IP addresses , choosing particular ports they are interested in. The famous *Zeus* botnet has been observed utilising this method, with the scanning only targeting specific ports commonly used for vulnerable web servers across multiple hosts simultaneously [41]. The *vertical* scanning technique involves scanning a single host for a range of different ports. This is often used to enumerate all services that are open on a host, uncovering all possible attack vectors. While this may not be the most efficient method in which botnets scan large number of IPs, this is typically used in manual intrusion attempts and is incorporated in the popular Nmap tool [42].

An emerging avenue in which botnets have been able to generate revenue is the mining of cryptocurrency. This is widely known within the cyber security community as *cryptojacking*. A recent report by the European Union Agency For Cybersecurity (ENISA) documents that 61.4 million cryptojacking attacks were identified within 2019 [43]. This type of attack leverages the computational power of a large number of infiltrated servers to generate substantial and untraceable income. The report lists a variety of legitimate mining software used by botnets to perform this function. In addition, the report identifies Monero as being the most popular cryptocurrency to mine by various botnets.

2.1.1.2 Ransomware

Ransomware, much like botnets, employ vast scanning and exploitation campaigns. However, in this case, the objective is very specific: the encryption of sensitive files on an infected system. The malware leaves ransom notes detailing how to transfer payment, typically in the form of untraceable cryptocurrency or gift cards. In return, operators of the attack communicate the decryption key to the victims, allowing them access to their files once again. Holding data for ransom has been around for many years. Before the inception of Bitcoin, traditional payment methods typically did not provide a layer of anonymity and came with some risk for attackers. However, the widespread use of cryptocurrency has led to the creation of an explosive wave of innovative ransomware families being authored, each with various tactics and techniques used for infiltration and encryption schemes [44]. The most publicised ransomware attack occurred in 2017, through powerful exploitation of SMB services. WannaCry ransomware infected thousands of devices in short succession, making use of powerful self-replication technique [45]. As a result, the UK's National Health Service (NHS) suffered several breaches, where computers in general practices were targeted and patient records were encrypted, leaving doctors and nurses to work without adequate resources [46]. This could have been easily prevented as the exploits used in WannaCry were EternalBlue and EternalRomance, exploits developed by the National Security Agency (NSA) and subsequently leaked by malicious actors. The exploits target various flaws in Microsoft's implementation of the SMB protocol, and grant system level privileges to attackers. These exploits are typically utilised alongside the DoublePulsar implant, a backdoor developed by the NSA. This enables custom commands to be received through hooking the SrvTransactionNotImplemented method within the SMB service [47]. A recent investigation revealed through mass scanning of the Internet that 56,586 hosts were infected with the DoublePulsar implant [48]. Interestingly, there is no authentication to use the backdoor. As such, WannaCry also scans and actively exploits targets already infected with the DoublePulsar implant. Prior to these exploits being leaked, Microsoft released patch MS17-010 fixing the vulnerabilities. However in the case of the NHS and other enterprise networks, they did not manage to patch the systems in a timely manner, leaving them vulnerable and highlighting severe security problems in major institutions [49].

Ransomware employs either symmetric encryption, asymmetric encryption, or a combination of both. Poor implementation of symmetric key encryption methods have lead to developments in decryption tools, circumventing the ransom payment and decrypting the files through identification of symmetric keys used in the encryption process. TeslaCrypt is an example of one such ransomware family, utilising AES-256 (Advanced Encryption Standard) as the method of encryption and storing symmetric keys on the disk in a file named 'key.dat' in the users application data folder. As a result, decryption is trivial, with Cisco releasing a tool of their own which restores all files destroyed by TeslaCrypt [50].

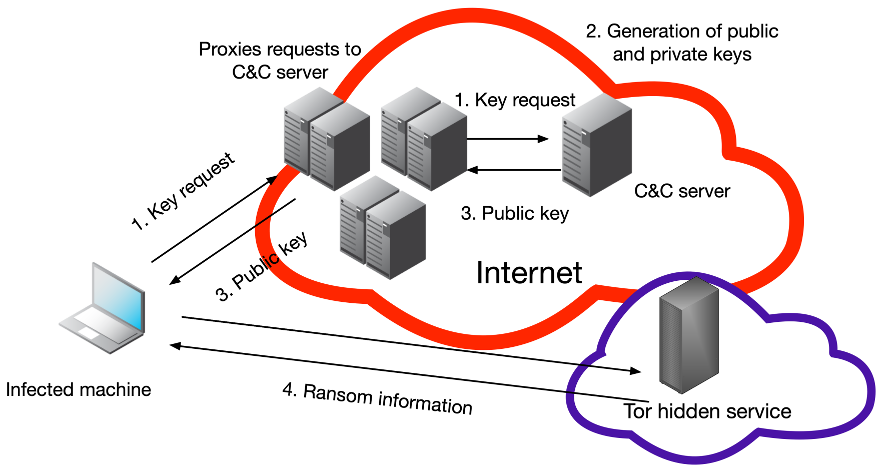


Figure 2.1: Asymmetric ransomware encryption process.

A more robust approach is the use of unique asymmetric keys for each victim. This process is depicted in Figure 2.1. Upon successful infiltration of systems, communication is initiated with the C&C servers, typically using some form of additional encryption to prevent interception of data, where the C&C server generates private and public keys unique for the victim. This process is stage

1 within the figure. The C&C server will then log victim system details, along with encryption keys to an internal database and send only the *public* key to the victim. This public key is then used in the file encryption process on the victim system. This process is stage 3 within the figure. Finally, the victim will be informed of his misfortune and a ransom note will be presented, as highlighted within stage 4 of the figure.

As asymmetric key encryption is much slower than symmetric key encryption. Authors combine the two to gain benefits in speed and security. Taking CryptoLocker as an example, this family creates a symmetric AES key to encrypt files on the host. Once this process is complete, the public key received from the C&C is used to encrypt the symmetric key, saving it to disk along with the encrypted files, ensuring the original encryption key is erased from memory. Thus, only the private key on the C&C server can decrypt the symmetric key, requiring ransom payments to be made [51]

If there is no connection to the Internet on the infected device, this method can not be used as communication with C&C servers cannot happen. In this case, some ransomware families still encrypt the files, typically using AES-256 with varying Initialisation Vectors (IVs). Furthermore, they attempt to erase the symmetric key used from memory and disk, along with deleting shadow copies to ensure backups cannot be used to restore files. Ransom notes are still delivered to victims and payments can still be made, however the attackers do not have access to encryption keys, leaving victims helpless in this scenario despite making payment.

2.1.2 Multi-stage intrusion

Multi-stage intrusions are attacks which incorporate multiple techniques, tactics, and protocols, such as those discussed previously, to achieve a predetermined goal. In this scenario, attackers typically follow an attack life cycle model, which details the steps required by an adversary to infiltrate computer systems. For example, after the successful infiltration into infrastructure through phishing attempts, attackers may execute malicious code in order to further establish a foothold into infrastructure through lateral movement and additional exploitation of vulnerable services.

This model was first introduced by Lockheed Martin to aid analysts in understanding what information may be available for defensive courses of action, identifying different stages in which attackers leave traces of compromise [52]. This model is depicted in Figure 2.2, with each stage being required to be complete before leading to the subsequent stages.

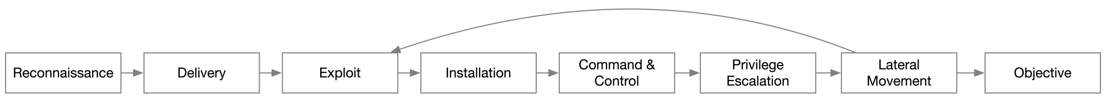


Figure 2.2: Attack life cycle model

Reconnaissance: Initially looking to gain as much information about the target as possible, attackers perform reconnaissance through identifying vulnerable technical components and key staff members within the organisation. In this stage, enumeration of computer systems and their respective IP addresses, domains, and services is undertaken to identify potential vulnerabilities and attack vectors. The aforementioned vertical scanning technique is often used to enumerate potential services to exploit, utilising Nmap to perform stealth TCP scans. In the case of the discovery of web servers, publicly available tools such as gobuster [53] can be utilised to bruteforce directories and files, DNS subdomains, and virtual hosts in an attempt to discover hidden services which do not follow proper security protocol. Furthermore, contextual information about the target is also gathered in this stage, possibly identifying members of the organisations through Open Source Intelligence (OSINT) tools. OSINT is a form of intelligence collection through publicly available sources on the Internet. Social media is often examined as part of this process. Through the identification of employee profiles, email addresses, usernames, and organisational hierarchy can be ascertained. This discovery can potentially lead to future attack vectors, with spear phishing attempts being enhanced with convincing contextual employee information. Moreover, usernames and email addresses could be leveraged as credentials for accessing computer systems, with their corresponding passwords also potentially being able to be recovered from password dumps of leaked databases [54].

Delivery: Based on the attack vectors identified through reconnaissance, delivery of the attack payload is then processed. This step can also be approached through multiple avenues, with spear phishing and watering hole attacks being the most common [55]. Spear phishing relates to intricately crafted phishing emails targeting entities identified to be of significance in the organisation with malicious attachments or links. The concept of watering hole attacks follows an analogy of a predator patiently waiting at a watering hole, biding its time knowing that eventually the victim will fall prey and enter the watering hole. Similarly, attackers identify websites frequented by the victim and attempt to infect that website through such methods as XSS (Cross Site Scripting). Consequently, when the victim loads the website, they also become infected.

Exploitation: The exploitation stage seeks to activate the payload delivered in the previous stage, gaining initial access into the target network. This is typically achieved through actors clicking attachments or malicious links in the aforementioned spear phishing methods, and also through the execution of malicious code on target computer through exploitation of vulnerable services.

Installation: Upon successful exploitation of systems, the next step involves the installation of malware to gain remote access and persistence through restarts.

Command & Control: Through the installation of malware, C&C mechanisms are established which specify how the malware operates, enabling further compromise.

Privilege Escalation: If the user account compromised within the host does not have sufficient privileges to perform necessary actions, for example read sensitive documents or execute specific binaries, privilege escalation techniques can be used to gain access to administrator accounts. At the time of writing, GTFOBins [56] maintains a list of 185 Linux applications that have the capability to perform this operation, showcasing a glaring security risk if permissions are not meticulously administered.

Lateral Movement: Once deemed that the compromised system has been fully explored, lateral movement can be used to pivot into other internally networked systems, expanding the overall control over the targeted organisation. In this stage, credentials harvested from previous stages in the attack can be used to remotely authenticate into internal systems. This is then repeated, compromising as many systems in succession as deemed necessary to complete the attackers objective.

Data Exfiltration: When the target data is discovered, exfiltration is performed using encrypted protocols to transfer it to a location the attacker controls.

Despite this attack cycle being representative of attacks occurring within contemporary networks, there exists some problems with the model. It has been criticised as being too stringent in nature, as the diversity of attacks currently propagating means that not all attacks follow the model precisely, potentially missing some stages if not necessary or applicable to the overall goal of the attacker [57]. For example, privilege escalation may not be necessary if the exploit stage grants root access, such as the aforementioned EternalBlue exploit. Moreover, lateral movement does not need to be utilised if the compromised system already contains the required data.

2.1.2.1 APTs

APTs (Advanced Persistent Threats) are cyber attacks executed by sophisticated and well-resourced actors pursuing access into government or other high-profile target infrastructure. These attacks often contain multiple zero-day vulnerabilities estimated to be worth in excess of hundreds of thousands of pounds. It is therefore no surprise that monetary gain is often not the purpose of these attacks, with geopolitical instability speculated to be at the fore of the most cutting-edge. These attacks utilise multiple attack vectors to gain footholds into cyber infrastructure and perform their objectives. They are described in [58] as having the ability to pursue objectives repeatedly over an extended period of time, adapting to defenders resistive efforts , and having specific targets. There are multiple phases within APTs, and as such there are models which attempt to classify the life cycle.

2.1.3 Service Disruption

Another attack method which does *not* require exploitation of services and infiltration of infrastructure is the Denial of Service (DoS) attack. DoS attacks attempt to disrupt the legitimate use of computer systems by flooding them with a large amount of traffic. This is done in an attempt to fully consume the resources of the system, leaving none to serve legitimate requests. Investigations have revealed the most popular type of DoS attack is the TCP SYN method [59]. This type of attack uses the SYN flag within the TCP header to indicate a connection should be opened to the client. Using source address spoofing, a single client can open numerous unused connections to the same server. These open connections consume resources on the server, and if enough are kept open, no more connections are able to be made to the victim server.

As discussed in the previous chapter, DDoS attacks involve the orchestration of multiple DoS attacks. In order to perform this type of attack, attackers first need to gain access to a large number of computers through exploitation of services [60]. As mentioned previously, this accumulation of infected computers is known as a botnet. Bots within the botnet coordinate with each other to perform DDoS attacks at massive scale. A recent report has identified that during June 2020, DDoS attack traffic was traversing the Internet at a rate of 2.8 terabytes per second [61]. As is often the case for cyber attacks, monetary gain is a motivating factor for DDoS attacks. Akamai published a report which outlines DDoS attacks occurring in August 2020 as a result of extortion attempts [62]. Not dissimilar in nature to ransomware, the threat actors typically communicate with organisations beforehand, warning of impending DDoS attacks unless a ransom is paid in bitcoin.

2.2 Cyber Threat Intelligence

As identified in the previous section, the evolving threat landscape consists of innovative and large scale attacks. The monitoring of these attacks is essential to reduce the rate at which they propagate, a process which utilises sensors scattered around the Internet to observe cutting edge adversarial methodology. There exists substantial research into this area, with contributions incorporating the deployment of honeypots and dark IP address space to capture unsolicited traffic. Moreover, analysis of malware which was used as part of the attack process is also undertaken in literature to gain insight into novel exploit techniques and the structure of C&C architecture. The information derived from these sources is known as CTI, which in contemporary studies has been used to profile malicious activity and enhance security mechanisms.

In this section, an examination of how threat intelligence aids understanding of emerging attacks is undertaken, providing details of how attacker methodology can bolster existing security mechanisms. This is initially discussed in Section 2.2.1 by considering honeypots, systems which attract attacks through deception. When monitoring emerging threats, it is crucial to correlate data with external sources to corroborate suspicion of illicit activity. Cyber Threat Intelligence Services (CTIS), as described in Section 2.2.2, provide such functionality.

Serving as additional data points, these services have the capability to confirm whether attacks identified locally are symptomatic of a larger campaign orchestrated throughout the Internet, further augmenting knowledge of threat actors through contextual information.

2.2.1 Honeypot

Honeypots are systems under observation which contain components that masquerade as legitimate enterprise infrastructure in order to catch unsuspecting adversaries leveraging previously unobserved exploits, attack tactics and patterns used for infiltration [16]. Utilising these honeypots grants unrestricted access to emerging CTI through extracted log data, which is otherwise extremely difficult for the wider community to access. This is in part due to an anonymisation process, which data is often subject to before public release. In many cases this removes information relevant in the analysis of attacks, such as IP addresses. In addition, cutting edge attack data is difficult to obtain due to deficiencies in research and secrecy from industry.

The benefits of this approach relate to the fact that activity relating to these systems is likely to be malicious as all communication is unsolicited. Therefore, honeypots are an invaluable asset in the identification of novel attack vectors. The types of attacks honeypots encounter are vast, as the number of different services emulated by honeypots grow, diverse attack vectors can be captured and discovered, leading to the generation of threat intelligence data. Ghourabi et al. characterise telemetry observed from their bespoke web application honeypot, and identify that among benign Internet traffic there existed exploitation attempts such as Structured Query Language (SQL) and XSS injection, fuzzing penetration testing tools, and scanning attempts [63]. The nature of the deployed honeypots dictate the types of attacks encountered, with honeypots deploying a large number of different services granting observation of more diverse types of attacks. Yahya et al. [64] deploy a range of emerging honeypots that emulate a large number of popular services. Through the analysis of the captured telemetry, the authors identify diverse attack techniques such as DDoS attacks, bruteforce attempts, and exploitation of vulnerabilities targeting various software implementations.

2.2.1.1 Honeypot Types

As a honeypot is a conceptual method of observing attackers, there exists multiple varying implementations with each providing different functionality. Despite this, honeypots can be loosely classified into three distinct types: low-interaction, medium-interaction, and high-interaction. A low-interaction honeypot typically exposes fake services with no underlying layer in which adversaries can exploit and propagate further through the network. These are mainly used to gather metrics relating to credentials and connection information gathered from headers within packers. Examples of these include Glutton [65] and Heralding [66]. As a result, these types of honeypots typically gather a lot of scanning data, due to attackers concluding that there is no attack surface and therefore will not attempt any further probing or exploitation.

Medium-interaction honeypots aim to masquerade as a realistic service such as File Transfer Protocol (FTP) and Secure Shell (SSH) to provide an underlying emulated environment which allows attackers to further interact and capture exploits without any real risk of infiltration. An example of such a service is Cowrie [67], an SSH and Telnet honeypot capable of capturing brute-force attempts, command-line input, and malware dropped within the shell session. Another prominent medium-interaction honeypot is Dionaea [68]. Dionaea emulates multiple services spanning both UDP and TCP, with the ultimate intention to trap malware exploiting vulnerabilities in services exposed to a network. This honeypot leverages libemu [69], an x86 emulation framework, to execute exploits crafted by attackers in order to gather shell code and other behavioural information.

High-interaction honeypots further add complexity as they deploy real operating systems and applications in which an attacker has the ability to completely gain access. While the risk of infiltration and propagation is much greater, the data gathered grants a system level view of operations, which further aids understanding of the extent of their behaviour. Furthermore, this is the only approach which grants insight into the entire chain of events, from initial reconnaissance to data exfiltration. The virtualisation of systems is a popular approach in deploying high-interaction honeypots, with hypervisors and containers being utilised heavily to prevent full takeover. Valicek et al. [70] take this approach through the cloud based deployment of remote high-interaction honeypots composed with Docker containers. This was made feasible through recent Windows 10 patches, which introduced native Windows Docker containers. Attackers were presented with an isolated server environment where they can perform any actions they wish, uncovering attack properties in the process. After the attack was deemed to be complete, the corresponding logs can be extracted and Docker containers reset to their original state, ready for another adversary.

2.2.1.2 Telemetry Extraction

There are typically three methods of extracting information from high interaction honeypots, namely: agent based approach, Man in The Middle (MiTM) proxy, and Virtual Machine Introspection (VMI). Sentanos et al. [71] identifies MiTM as a stealthy method in which collection of shell commands injected into an active SSH occurs. While providing common commands used to propagate malicious activity, this approach lacks the high level understanding of Tactics, Techniques, and Procedures (TPPs) used to analyse how these commands interact with the operating system and perform malicious actions. Adopting an agent based approach grants rich insight into operating system and application behaviour by emitting event logs which can be utilised for analysis of malicious actions at a host level. However, this approach can easily be deceived by knowledgeable adversaries. As the agent resides with the system itself, attackers can modify the data collection process to produce benign activity. VMI also has the ability to provide rich systems level information, and has additional benefits of complete isolation between the system under analysis and monitoring system, ensuring a complete and untainted view of systems operations. Sentanos et al. take this approach in their research, monitoring system calls of processes to gain an overview of operations within the system.

In the case of medium and low interaction honeypots, network telemetry is used to identify attacks. In addition to the logging capabilities provided by these honeypots, the raw network traces of attacks can be captured in order to derive a fuller understanding. From these raw network traces, essential statistical features can be derived for the inclusion in intrusion detection data sets [18]. At a software level, network traffic can be captured using the open source *Libpcap* [72] library, which offers an interface for capturing link-layers frames. This library also defines a standard format in which the captured frames are stored, known as the *tcpdump* format [73]. Sperotto et al. utilise this approach when capturing network traffic from honeypots. In their research, the authors create a new intrusion detection data set derived from this data capture.

In recent years, NetFlow has emerged as a popular method of monitoring network activity. This method solely collects flow based features which are derived from network traffic. As raw packet payloads are not considered, this method helps cope with growing scalability concerns introduced by expanding networks and increased speeds. Flow enabled devices are now offered by many vendors, such as Cisco supplying routers with NetFlow capabilites. Husák et al. utilise NetFlow to provide a flow based honeypot monitoring solution [74]. In their approach, the authors use NetFlow probes located at egress points of the network to gather data. The use of a NetFlow collector is also required in this scenario to further collect and analyse the captured data.

2.2.1.3 Deployment Locations

When deploying honeypots into a network, consideration of deployment location is required to produce desirable results. There are typically three locations suitable for enterprise network deployment, namely within Local Area Network (LAN), Demilitarized Zone (DMZ), or Wide Area Network (WAN) regions [16]. LAN refers to a collection of networked devices in the same physical location. A DMZ is a network which exposes external-facing services to an untrusted network such as the Internet. A WAN is a type of large network which connects devices over large distances. When positioning a honeypot within the LAN, it becomes integrated with production services and internal hosts. This approach tends to observe attacks emanating from within the network as opposed from the Internet. As a result, targeted attacks and malware looking to move laterally within an organisations network are often gathered from these honeypots. As interaction with honeypots within these local networks is unsolicited, it is highly likely that there is malicious intent. Therefore, these types of honeypots have been heavily utilised in production environments to provide a layer of defense against insider attacks [75]. One way of achieving this is through the use of honey tokens. These represent artificially crafted data which are unique to an individual and enable the monitoring of data movement. These act as bait for attackers, deceiving them into thinking that they have valuable data while constantly being tracked.

When honeypots are accessible from the Internet, the number of observed attacks rises dramatically. Attackers are constantly launching scanning campaigns, looking to identify vulnerable systems that they can compromise. The growth of Internet device search engines such as Shodan [17] has exacerbated this problem, allowing attackers to find all devices vulnerable to a particular exploit on the Internet in seconds. Due to this, honeypots which are accessible from the Internet are able to capture ample amounts of emerging attack telemetry. Emanating from various parts of the world, the attacks uncover novel techniques and tactics derived from the telemetry to gain an understanding of threats which are currently targeting infrastructure at scale.

2.2.1.4 Security Implementations

While honeypots do not directly increase security in a network through simple deployment, the analysis of the collected records can be leveraged in order to aid understanding of the threats targeting the network. This in turn can then be used to aid security mechanisms such as IDS or IPS. The approach taken in [76] leveraged high and medium interaction honeypots to produce host based intrusion signatures suitable for misuse-based Host-based Intrusion Detection Systems (HIDSs). The research outlines how deviation from normal behaviour observed within honeypots is detected, and how the corresponding signatures incorporating malicious actions are generated and can be used to prevent future compromise. Vasilomanolakis et al. [77] used a similar approach, through the deployment of bespoke Industrial Control System (ICS) honeypots in order to generate signatures of multi stage attacks by modeling each disparate protocol from the same host as a separate stage in the attack. For each of these stages, a signature is generated based upon characteristics of the network packet involved in the attack, which is then used by Bro IDS [78] to evaluate the detection capabilities.

Miyamoto et al. [79] adopt a different approach to aiding security within networks through the use of honeypots, namely through the migration of suspect adversaries from production services to honeypots. They forward suspect attacks identified by their Web Application Firewall (WAF) to VMs masquerading as legitimate enterprise services, ensuring that the attacker has no capabilities to target actual production systems. This approach performs live migration of VMs to achieve a perfect copy of environments for web applications, deceiving attackers in the process and wasting their time and efforts.

Matin et al. propose a detection approach for malware using honeypots and machine learning algorithms [80]. In their approach, the authors utilise the Endgame Malware BEnchmark for Research (EMBER) [81] data set that contains static analysis statistics of over one million Windows Portable Executable (PE) malware samples to train machine learning based classification algorithms. In their architecture, they outline the integration of honeypots to collect suspect executable files, and subsequently use the classification algorithms to identify the various families of collected malware samples. If a file is found to be malicious, the network administrator is notified, alerting them to the discovery of malicious actions.

Alata et al. predict the occurrence of new waves of attack on a given platform based upon the history of attacks for the given platform through the analysis and application of linear models on data collected from honeypots of varying interaction levels [82]. The authors outline the Collection and Analysis of Data from Honeypots (CADHo) project, an initiative which deployed distributed honeypots in many physical locations on five distinct continents in order to collect telemetry of emerging attacks. The authors also apply the use of statistical and probabilistic analysis techniques to further model the observed phenomena. Through the use of linear regression, the authors plot and predict trends inherent within the data, forecasting the number of attacks from different countries in the process.

Another popular use of honeypots in contemporary literature is the creation of IDS data sets. To evaluate the effectiveness of attack detection techniques they must be tested against data which includes benign and attack traffic. Such data is difficult to collect, with privacy and legal concerns making telemetry unavailable to the wider community. Honeypots gather a wide range of malicious and benign activity, representative of a portion of attacks currently propagating throughout the Internet. Therefore, telemetry derived from honeypots offer realistic insight into emerging attack processes and can heavily aid the security community with the evaluation of detection schemes. Many contemporary IDS data sets utilise honeypots to gather such data, and as such, they are investigated further in the section 2.3.5 of this thesis.

2.2.2 Cyber Threat Intelligence Service

CTI services provide organisations insight into relevant threat actors which aim to infiltrate infrastructure. These types of services are typically used in order to cope with the growing complexity of the cyber threat landscape and the increasing frequency at which cyber attacks take place [83]. Furthermore, these services all function in a bespoke manner, and deliver various types of intelligence. These range from services which systemically scan and profile the entire internet address range, e.g. Shodan, to services which maintain historical records of identified attackers in blocklists such as Maltiverse[1]. Due to commercial reasons, the true nature behind how these services operate is not divulged to the general public. Despite this, the intelligence offered by these platforms is an invaluable resource in the analysis and defense of emerging threats.

2.2.2.1 Services

Shodan is one of most popular CTI services, and boasts being the world's first search engine for Internet-connected devices. This service enables the comprehensive discovery of servers which contain a public IP address. Through entering various search terms into Shodan, a list of vulnerable servers is able to be produced. This indicates possible servers which have already been exploited, and any communication involving these servers should be treated as suspicious. Furthermore, this enables an understanding of the total number of hosts susceptible to various new vulnerabilities identified by researchers. This can be used to further assess the severity of the impact these vulnerabilities can cause.

Another such service which performs profiling of the Internet is Greynoise[2]. This service claims to maintain a large number of ephemeral servers in hundreds of data centers across the world, and perform omni-directional scanning to uncover properties about the topology of the Internet.

Maltiverse is another example of a CTI service. This service compiles a collection of various blocklists which detail IoCs. These IoCs include IP addresses, file hashes, and more. This can be used to understand whether potential communication attempts have been initiated by a known attacker contained within a blocklist. Another use case for this service involves checking against a list of

[1]https://maltiverse.com/
[2]https://greynoise.com/

known malicious files before execution. The benefit of using this service relates to the fact that it contains multiple blocklists, and provides a unified API to retrieve details from all of them at once.

2.2.2.2 Sharing Methodology

In order to enable the automated retrieval and processing of vast amounts of CTI data, these services typically provide a method in which the data is easily shared between parties. Standardisation of both the exchange mechanism and the format in which the data is exchanged has taken place to simplify this process and support a wide range of use cases.

In detail, the Structured Threat Information eXpression (STIX)[3] is a standardised language for structured cyber threat information representation [83]. The STIX language comprises of a large set of CTI classes, including indicators of compromise, TPPs, exploit targets, and threat actors. This enables a comprehensive understanding of suspicious cyber events.

To securely and automatically exchange CTI across organisations, the Trusted Automated eXchange of Indicator Information (TAXII)[4] was created. Increasing situational awareness of emerging threats, TAXII can be used to ingest large amounts of cyber intelligence represented in STIX format. A number of sharing mechanisms are provided by TAXII, and include peer-to-peer and source-subscriber models.

Despite the standardisation of exchange mechanism and intelligence representation, a large number of CTI services, including those discussed in Section 2.2.2.1, provide data through a RESTful API. While this also enables programmatic collection and analysis, the format in which the data is stored varies depending upon the CTI service. For example, Shodan grants access to its' intelligence platform through the RESTful API, and documents the data held within it in JSON format. The manner in which the data is represented in Shodan's JSON format further differs from that of other CTI services, meaning that applications which use multiple services are required to parse the intelligence using varying means.

[3]http://stix.mitre.org
[4]http://taxii.mitre.org

2.3 Intrusion Detection

An intrusion can be defined as any activity that causes damage to information systems [84]. An IDS is a system which identifies these malicious actions to ensure the security and integrity of computer systems is maintained [85]. These systems monitor various activities taking place, and if deemed to be malicious, alerts and remediation procedures are commenced to mitigate such threats. Data sources which describe this activity are diverse and manifest themselves in multiple ways, however, there are two broad classes of IDSs: Host-based IDS (HIDS) and Network-based IDS (NIDS). This distinction is based upon the type of input data and measurements used to detect abnormality.

However, in general, all types of IDSs contain three components: data collection, conversion of features, and a decision engine [86]. The data collection component ingests various types of data such as packets or system calls. To prepare the data for decision making, an IDS must convert the input data into a list of attributes called a feature vector. The decision engine consists of an algorithm which decides whether the input data is representative of malicious behaviour or not.

2.3.1 Network-based IDS

A NIDS monitors traffic traversing computer networks to detect malicious activity. The network traffic used to detect attacks is captured through various means, such as packet capture and NetFlow. Packets have traditionally been the method of choice for NIDS. However, due to the pervasive adoption of encryption, the packet payload is unable to be inspected. Additionally, full payload capture is computationally expensive and can lead to performance bottlenecks in high-speed networks [87]. Therefore, flow-based detection methods have become popular in literature as they encompass measurements from multiple packets in a connection and only analyse the packet header, not the payload.

2.3.2 Host-based IDS

A HIDS leverages data which originates from the system itself to defend against malicious actions. Multiple types of input data can be used to inform a HIDS about potential attacks, including system calls, system logs, file systems, Windows Registry, and processes. Notably, this method enables the detection of threats which do not manifest themselves over the network.

System call data is one of the most popular choices for use in HIDS. This has been suggested to be because there is no process which can obfuscate these events as they originate from the OS kernel, unlike the production of log files [86]. Modern operating systems are capable of producing hundreds of different types of system calls. The main drawback related to this approach to HIDS is the high computational overhead required for collection and processing.

2.3.3 Misuse Detection

The previous two sections categorise IDSs into groups based upon the data source. However, they can also be categorised based upon the method used to identify attacks. These groupings are known in literature as misuse-based intrusion detection and anomaly-based intrusion detection. Misuse-based IDSs typically leverage a database of known attack signatures to detect attacks. These systems use pattern matching to identify whether suspect activity is contained within the known malicious database. If the activity being compared matches with records within the existing database, an alert can be generated to notify administrators of malicious activity.

These types of systems generally grant excellent detection accuracy for known attacks. However, they struggle against emerging and novel threats, due to the fact that the signature database must be updated to reflect the new attacks. The increasing rate at which novel attack vectors are discovered and utilised have rendered this approach less effective [88].

2.3.4 Anomaly Detection

Anomaly detection approaches have drawn immense interest from the research community due to them alleviating challenges incurred by signature-based systems. Most notably, these include the ability to detect zero-day attacks as they do not require a known malicious signature database [89]. In this approach, a model of normal systems behaviour is created using machine learning, statistical-based or knowledge-based methods [84]. Any monitored activity which deviates from this normal behaviour profile is treated as an intrusion. However, these systems generally suffer from a greater false positive rate compared to signature alternatives. This is caused through various means, including a lack of strong ground truth within training data.

2.3.4.1 Machine Learning

Machine learning based methods have been shown to be adept in the detection of a variety of malicious activity. In this approach to anomaly detection, machine learning models are trained using data extracted from intrusion detection data sets. Their capacity to detect attacks is then evaluated using test data containing unknown records. There are many distinct types of machine learning algorithms that have been applied to intrusion detection scenarios, such as decision trees, clustering, neural networks, and genetic algorithms.

To further distinguish between machine learning techniques, there exists both supervised and unsupervised algorithms. Supervised algorithms require a ground truth in the form of target labels within training data. This approach attempts to create a function which correctly maps input data to the corresponding output, using observations learned through training. Supervised learning is typically achieved in regards to classification, which maps an input to clearly defined output labels. The model will always assume the output labels within training data are accurate. However, this is not always the case in real-world scenarios. Clearly, this highlights the importance of an accurate ground truth, as incorrect labels reduce the effectiveness of model prediction. Conversely, unsupervised learning techniques do not require a ground truth. These type of algorithms attempt to understand and predict based upon the natural structure of data points. However, unsupervised approaches often grant lesser accuracy when compared to supervised approaches [90].

Research conducted by Sangkatsanee et al. leveraged decision trees to detect a variety of malicious behaviour, including DDoS and probes, in an online fashion [91]. Their approach verifies the detection capabilities through real attacks orchestrated against victims, where the network traffic is captured and accurately classified. This type of evaluation ensures that the attack detection capabilities work within real-world scenarios. While successfully classifying primitive attacks with high accuracy, this approach does not consider attacks which are sophisticated and emerging in nature. This raises further questions regarding the appropriateness of such an approach to detect the myriad attacks which encompass the emerging threat landscape.

In another approach, Rahul et al. [92] construct a Deep Neural Network (DNN) to classify network traffic which provides excellent detection rates, granting around 93% accuracy when evaluated on the KDD data set. Despite the promising results obtained under decision making scenarios, they neglect to evaluate their system using a recent data set. The KDD data set is now over two decades old, and, as further explained later in this chapter, it has been argued by contemporary researchers that it does not reflect current network conditions or relevant attack patterns. Furthermore, the authors neglect to perform a performance evaluation for either the training or decision making processes. As a result, it is not clear whether such an approach is appropriate for the evaluation of live network data.

Morfino et al. [14] identify challenges related to the volume and velocity of data transferred through modern networks. Based upon this, they implement a near real-time intrusion detection mechanism which integrates with big data technologies. Their approach is able to detect DoS attacks orchestrated against IoT devices with high accuracy. Furthermore, they evaluate overall training time using varying number of data instances, demonstrating inherently low training times through the construction of various models. One aspect of their system which was not evaluated was the efficiency of the prediction process.

Lobato et al. implement a threat detection framework which leverages stream processing technology [93]. In order to address the "needle in a haystack" problem associated with the vast amount of data network providers must process to identify threats, the authors leverage Apache Storm, a distributed stream processing computation framework. The framework is evaluated using a bespoke data set which incorporates DoS attacks and scanning probes. The data set is not made publicly available. Tested using a handful of different classification algorithms, the framework is able to detect these primitive attacks with an accuracy of around 95%. Despite the promising performance and accuracy granted by this framework, it is not evaluated against threats of a sophisticated nature. Therefore, it is unknown how effective it would be in contemporary real-world scenarios.

Viegas et al. [94] create a machine learning based intrusion detection framework for high speed networks, BigFlow. BigFlow leverages the stream computation framework, Apache Flink [95], to accomplish this. The authors also acknowledge the requirement of realistic training data for the accurate detection of anomalies. To address this, they make promising steps towards the practical integration of automated data set creation and anomaly detection for the identification of cyber attacks. They implement a feature extraction module which takes raw network packet captures from the MAWI archive [96] and creates flow-based features to compose a novel data set, *MAWIFlow*. Despite including the capability to engineer features, this framework relies on an external, extremely dated and problematic labelling methodology [97], resulting in a questionable ground truth for their data set.

Dong et al. [5] implement a NIDS on top of the distributed processing framework, Apache Flink [95]. They acknowledge the problems relating to current big data solutions typically performing batch processing, and instead leverage Apache Flink's streaming capabilities coupled with Apache Kafka's [98] distributed message bus to detect attacks in an online fashion. Using a DNN, they are able to detect various attacks with around 94% accuracy. Despite the promising detection results, they neglect to perform a performance evaluation to assess the benefits granted via integration with the aforementioned big data technologies. In addition, the authors evaluate their system using the KDD data set, as discussed in Section 2.3.5, it has multiple limitations.

Rathore at al. [99] propose an IDS which is reported to be suitable for high-speed networks. They leverage the Apache Hadoop [10] software library to perform distributed processing of network telemetry. While the system is able to process streaming packets from network devices, such as switches, it does not use a distributed message bus. The system is also only evaluated on the KDD data set, which poses questions about the validity of the detection mechanism.

2.3.5 IDS Data Sets

In order to evaluate the effectiveness of an IDS, the accuracy of the attack detection procedure is typically used. This process typically assesses the FP (False Positive) and TP (True Positive) rates of detection. A *data set* which contains both benign and attack traces is necessary to assess this metric. Currently, there still exists challenges in the collection and release of this data to the public. Most notably, privacy concerns have been raised over personally identifiable information, leading to heavy anonymisation in most contemporary IDS data sets [100]. Recent studies have shown that the most widely adopted data sets are heavily dated and no longer reflect modern attack vectors. Moreover, investigations have also discovered other deficiencies relating to lack of traffic diversity, attack techniques, and inappropriate features, which can affect the transparency of the IDS evaluation [101].

In the security community there is a lack of public IDS data sets, with current literature urgently appealing for high quality labelled attack data [102, 103, 104, 22, 7, 105, 106, 107, 108, 109]. An IDS data set of high quality must include a comprehensive reflection of contemporary threats and a range of benign behaviour spanning multiple protocols, hosts, and applications [101]. Furthermore, an often overlooked aspect is the methodology behind labelling of the data set. Correct labelling ensures all observed attacks are identified, thus dictating the reliable outcome of any IDS [110]. Additionally, labelling enables the use of supervised machine learning algorithms, which have been shown to usually provide better detection accuracy than their unsupervised alternatives [90]. A study conducted by Abt et al. discovered that the majority of publicly available data sets are not labelled, assigning the cause to be due to the labour intensive nature of manual labelling of data sets [23]. The authors then argue that the *"missing labelled data problem"* affects repeatability and comparability of research.

Recently, there have emerged data sets that have been released to the public that utilise novel automated mechanisms to alleviate this problem. This section discusses the various ways this is embodied in modern literature, as well as highlighting popular publicly available data sets that use methods designed to provide a ground truth in the form of target labels, alongside other techniques used to compile data sets whilst simultaneously highlighting the deficiencies in their approach.

2.3.5.1 Data Set Labelling

In order to successfully label a data set, the *ground truth* must be discovered. In literature, there are only a handful of public data sets which consider the ground truths and provide corresponding labels. This has traditionally been examined and implemented using labour intensive methods of manual analysis. The KDD '99 [19] ground truth is an example of a data set incorporating this approach.

Despite this data set being over two decades old, it is still considered a milestone in the community because of the accuracy of the provided labels [97]. As a result, this data set has been exhaustively used in literature to evaluate IDS, despite the fact that the modern threat landscape has shifted significantly, with many researchers suggesting experiments performed with this data set are no longer relevant [22]. Since the public release of KDD '99, novel approaches that provide automated routines to derive a ground truth for intrusion detection data sets have been outlined in literature.

One of the first data sets to incorporate such an approach was the Kyoto 2006+ data set [18]. In their approach, honeypots were deployed to attract malicious intrusion attempts. Kyoto 2006+ provides data with partial labels such as to enable the separation of normal and malicious traffic. Unlike the KDD data set, the labels do not describe explicit types of attacks. Much attention has been given to enhancing classification performance using data with partial labels, such as [111, 112], due to the lower effort required to provide such labels. This is also true in network security, since specifying the distinct attack description is often more challenging. In this case, Kyoto 2006+ provides labels by assigning traffic relating to honeypots deployed within their architecture as malicious. Consequently, this approach does not consider legitimate traffic involving honeypots.

As identified by Sperotto et al. in [113], not all telemetry associated with honeypots are attacks and malicious in nature. In their work, they identify a number of non-malicious traffic to and from honeypots which they consider "side effects". It has also been identified that honeypots are also targeted by legitimate services which actively probe and document internet connected devices, such as Shodan [24]. In order to differentiate between actual attacks and normal traffic relating to honeypots, a strong ground truth is required.

As a result of their investigation, Sperotto et al. also create the first flow based intrusion detection data set and additionally provide labels through automated routines. In their approach, the authors generated labels through the correlation of alerts generated from honeypot logs and network flows derived from traffic capture. This is implemented in their work by an algorithm which matches alerts from honeypots to their corresponding network traces based on their timestamps. Based upon the category of the alert generated on the honeypot, the corresponding network traces were able to be labelled.

In more recent embodiments, unsupervised anomaly detection algorithms have been utilised to derive a ground truth for an intrusion detection data set. This is the approach taken in the MAWILab data set [114], where they used automatic labelling mechanisms to produce a continuous stream of labelled telemetry. The data set consists of labelled network traces gathered on a trans-Pacific link between the United States and Japan. However, the MAWILab ground truth is questionable since it has been derived through the combined output of four seperate decade old unsupervised network anomaly detectors, and upon manual inspection some anomalous flows exhibit benign characteristics [97, 114].

The approach taken by Aparicio-Navarro et al. also utilises unsupervised anomaly detection techniques to automatically label a data set [115]. The authors gathered benign telemetry from a wireless Access Point (AP), and injected malicious activity through the use of the AirPwn [116] tool. From this data, they present automatic routines which generate three levels of belief for each analysed frame using Dempster-Shafer theory. These are beliefs in normal, which indicates how strong the belief is that the frame is benign, belief in attack, which indicates how strong the belief is that the frame is malicious, and belief in uncertainty, which measures how doubtful the system is in deciding whether the frame is benign or malicious. Based upon the values for these beliefs, the authors then define a threshold between the belief in normal and the belief in attack, which enables misclassified instances to be discarded. Finally, the data set generated by the authors is not intended to train supervised IDSs. Instead, their approach is used in tasks of feature selection.

The approach taken by B-IDS [117] leverages one-class Support Vector Machine (SVM) and a Rival Penalized Competitive Learning (RPCL) network to develop attack ground truth. These algorithms produce a classification outcome, which is combined with the Dempster-Shafer theory such as to produce a singular output. Via this approach, the authors are able to distinguish between normal and attack packets extracted from real traffic traces with an error rate of around 2%.

2.3.5.2 KDD '99 Data Set

As previously mentioned, the KDD '99 data set is one of the first publicly available IDS data sets and remains the most widely adopted in the evaluation of anomaly detection methods [104]. In 1998, researchers at Lincoln Laboratories in MIT University generated the DARPA98 data set to perform thorough investigation of IDSs. The data set incorporates results from a simulation containing both normal and abnormal traffic collected from inside a military network. The simulation concluded after nine weeks, resulting in five million connection records included in training data and two million collection records included in the test data. The KDD data set emerged from the tcpdump portion of DARPA98 traces, incorporating rich features derived from extensive analysis.

The data set contains over 40 unique features for each connection, incorporating both host measurements, such as the number as files accessed, number of failed logins, as well as standard network measurements such as flow duration and number of bytes sent to destination. The attacks embedded in the data set were based on modified scripts gathered from various sources, and were additionally synthetically injected into the background traffic. As identified by McHugh, no attempt was made to ensure that the synthetic attacks were realistically distributed in the background noise [118]. Furthermore, Tavallaee et al. claims an inherent problem within the data set is that the workload of the synthesized data does not reflect traffic in real networks [119]. These problems highlight the challenges incurred when utilising synthetic data in the generation of IDS data sets, heralding innovation in order to capture a high quality realistic data set. The total number of attacks synthetically injected into the data set was over 300 and are categorised into the following: DoS, User to Root (U2R), Root to Local (R2L), and Probing. The labels of the attacks were said to be ”verified by hand” and a very labour intensive process, however the details of the verification process are not divulged [120].

While this data set has been heavily utilised in literature, several researchers have reported major problems inherent within the data, which can cause issues in the evaluation of IDSs [110, 121]. The main problem relates to the difference in probability distributions of the training and test set. This is due to some attacks injected solely into the test set, which leads to bias in classification methods [101]. Another problem is the redundant data within the data set. Investigations have discovered up to 75% of the records are duplicated, causing additional bias to learning algorithms towards more frequent records, diminishing the ability to learn from infrequent records such as the U2R attacks [119]. As a resolution to these problems, the NSL-KDD data set was produced from the KDD data set, which aimed to address some of the inherent shortcomings [100, 119]. However, each of these data sets are extremely dated, with the attacks no longer being relevant to what is experienced today. As a result, they cannot serve as a viable data set anymore, with studies stating that their only role in contemporary research is to test whether an approach is hopelessly broken [22].

2.3.5.3 MAWILab Data Set

MAWILab is a database containing approximately 500 billion packets captured at an intercontinental link between USA and Japan. Every day since 2001, this database is updated with a 15 minute trace of traffic in packet-based format [114]. The updated nature of this data set is the first of its' kind, enabling an evolving understanding of normal network conditions as well as anomalies.

Despite this, the data set has a number of limitations. To begin, the traffic traces are only available in tcpdump [122] format. This inhibits the usability of this data set, since there must be intermediary steps taken, such as engineering valuable features and converting to Comma-Separated Values (CSV) file, in order to analyse using modern techniques. In addition, the labelling mechanism leveraged to distinguish between normal and anomalies within the captured traces, consists of the combination of four dated unsupervised machine learning techniques. This has been heavily criticised within literature [97, 114], and as a result the data set is not widely adopted. This highlights the importance of a robust labelling method.

2.3.5.4 Kyoto 2006+ Data Set

The Kyoto 2006+ data set was created by researchers at the National Institute of Information and Communications Technology through the collection of attacks and benign telemetry spanning from November 2006 until August 2009 [18]. In their approach, Song et al. utilise honeypots as an attack capture medium, deploying various enticing systems which lure adversaries into revealing their techniques. In the same network, servers conducting benign tasks such as mailing service and DNS server were also deployed to generate realistic background traffic resembling legitimate enterprise infrastructure. The way in which Song et al. label the data set is dubious, as they regard all telemetry gathered from the honeypots as malicious and all telemetry gathered from the mail and DNS server as benign. It is stated by the authors that all traffic related to the DNS and mail server was labelled normal as there were only a “few” attacks on them. However, the number of false positives, i.e., the traffic related to the honeypot which was not an attack is not divulged by the authors, leading to an inconclusive understanding of the accuracy in their approach.

Utilising Bro [78] to convert raw traffic into session data, twenty four relevant features were derived which represent each record. Of the twenty-four, fourteen were statistical and encapsulated the flow characteristics of the session. An additional ten features were derived which enable the effective investigation into what attacks happened within the data set including various detection metrics such as whether shellcode or malware was detected during the session.

2.3.5.5 ISCX 2012 Data Set

The ISCX 2012 data set [123] was created by researchers at the Information Security Centre of Excellence in the University of New Brunswick. This data set was created using a combination of attack and benign profiles. The benign profiles are created through the examination of typical non-malicious traffic extracted from Packet Capture (PCAP) files. The statistical distributions of the times traffic was produced were calculated from these observed traces, and were subsequently used to generate synthetic traffic representative of these observed distributions.

The attack profiles are generated using human knowledge and assistance in execution. In total, this data set contains only four attacks which are orchestrated manually against servers within their local network. Naturally, these are real attacks, however, they do not include the variety of exploits leveraged in the wild to infiltrate large numbers of servers remotely. As the attacks captured are not orchestrated by real threat actors, it can not be concretely concluded that they are fully representative of attack patterns for that period. With the evolution in network infrastructure and attack patterns, it is also questionable that this 8 year old data set is still relevant.

2.3.5.6 CTU-13 Data Set

The CTU-13 data set contains attack data consisting of 13 distinct scenarios in which botnet malware was executed [124]. This data set was authored by researchers at Czech Technical University with the intention of capturing normal and malicious traffic within a real networked environment. Normal traffic is labelled based upon certain filters outlined by the authors. Malicious traffic is labelled for traffic relating to known infected hosts. Background traffic is also included in the data set. However, this includes both normal and attack telemetry.

Due to the static nature of this data set, it is not able to fully reflect the evolving modern threat landscape. Furthermore, this data set only includes attack traces created through the execution of botnets. Whilst still a popular method of infection in recent times, there exist myriad other types of threats which wreak havoc on computer networks. Therefore, the data set should only be used to evaluate the performance of botnet detection mechanisms.

2.3.5.7 UNSW-NB15 Data Set

The UNSW-NB15 data set was synthetically generated by researchers at the Australian Centre for Cyber Security (ACCS) [101]. They utilised the IXIA traffic generator tool across three distinct servers, two of which produced benign traffic whilst one produced abnormal traffic. The data was collected over two periods, lasting 15 and 16 hours respectively. The duration of the collection is relatively short compared to other data sets, therefore there could exist inherent bias and lead to over fitting when employed alongside machine learning algorithms. Furthermore, the traffic was generated within a small network environment, with 45 distinct IP addresses contained within the data. The authors list nine different attack types which were synthetically injected into the data set including backdoors, exploits, DoS, and more. However, the authors do not provide details about any individual attack or their properties, which raises questions regarding whether the attacks are representative of attacks currently taking place.

The main drawback of their approach relates to the fact that all data is synthetically generated, thus, it is not reflective of real operations within enterprise networks. Moreover, the injected attacks are known before-hand, primitive in nature, and not representative of attacks currently propagating.

2.3.5.8 CICIDS2017 Data Set

The CICIDS2017 data set contains labelled network traces of benign and malicious activity conducted across twelve different servers [104]. Sharafaldin et al. construct a victim network consisting of heterogenous devices, with one port on the main switch configured as a mirror port to completely capture all traffic traversing the network. The benign behaviour and corresponding telemetry was generated using the B-Profile system [100], which emulates legitimate user behaviour through web-crawling and generating HTTP traffic. The authors then construct an attack network, in which attacks are orchestrated against the victim network. These attacks are categorised into six attack profiles, which were based on “the last updated list of common attack families”, and include brute force, DoS, DDoS, botnet, and infiltration attacks. The authors do not go into detail about how they are representative of attacks currently propagating.

To extract the network traffic features, the CICFlowMeter [125] tool is used. This tool extracts 80 flow based characteristics from a pcap file. The importance of each feature with regard to the target label is also investigated by the authors. Using the RandomForestRegressor module of scikit-learn, the best feature sets for each category of attack is outlined.

There are a number of shortcomings related to the CICIDS2017 data set. Despite injecting attacks into a victim network, it has been identified that in 288602 instances there are missing target labels [126]. Furthermore, this data set also contains very high class imbalance. High class imbalance arises when a class within the data set becomes a large majority, and in the case of classification using the data set, bias towards the majority class occurs [127, 128]. In the case of the CICIDS2017 data set, the majority class contains 83.34% of all records, whilst the minority class only contains 0.00039%. If a random sample is used for the training and testing of classification algorithms, it becomes likely that the minority class will not be included in training data. As a result, the classification algorithms will fail to recognise patterns of such attacks in future testing, as it has not previously been encountered.

2.4 Research Challenges

Through an investigation into background and related work as part of the research undertaken in this thesis, a number of deficiencies and gaps in research have been identified, which limit contributions into the cyber security domain. In detail, these include:

- **Research Challenge 1:** The research community lacks high-quality attack data.

As outlined in Section 2.3.5, the research community requires novel attack data to fully evaluate anomaly detection approaches to intrusion detection. Such data sets are scarcely released to the public, which motivates innovation to answer Research Question 1. When creating a data set, it is essential to identify certain properties that ensure it is of high quality and can be successfully leveraged by the researchers. Małowidzki et al. outline several of these properties in [107]. Included in this list of features which make a high quality data set is the aspect of realism. Realism, in regards to intrusion detection data sets, manifests itself in multiple ways. These include the inclusion of network traffic captured from actual networks instead of simulated network traffic through mathematical models [106, 129]. Furthermore, attack telemetry which is representative of real attacks currently propagating should also be incorporated. The research conducted by Hindy et al. surveys the available IDS data sets, and finds that data sets with dated attacks render IDSs ineffective against emerging attacks or zero days [20]. As the research conducted within this thesis aims to explore how emerging attacks can be identified using machine learning algorithms, a data set incorporating a range of emerging attack techniques is required to evaluate the effectiveness of this approach.

The availability of emerging attack telemetry in the form of a data set is limited, and as discussed in the related work, they all incur limitations, which inhibit their practical use in IDSs. Most notably, the KDD-99 data set is the most widely utilised in the evaluation of IDS in literature. However, due to the age and irrelevancy of the attacks incorporated within the data set, in modern real-world settings the detection capabilities achieved from training machine learning algorithms on this data is not acceptable [92].

The nature of honeypots enable an unrestricted view of malicious activity currently propagating over the Internet, capturing detailed accounts of attacks likely currently being encountered by enterprise networks. As a result, honeypots have been identified as a solution to the problems faced by data sets requiring realistic malicious activity [7]. Through the planned deployment of honeypots, attack telemetry can be gathered, which incorporates real emerging techniques and tactics used for mass infiltration. The analysis and processing of this telemetry is required to collate a robust IDS data set, which includes cutting edge malicious and non-malicious records.

- **Research Challenge 2:** There is an emphatic lack of ground truth with regards to publicly available intrusion detection data sets.

As previously discussed in Section 2.3.8, modern publicly available intrusion detection data sets largely omit the establishment and validation of a ground truth. Neglecting to establish a ground truth inhibits the utility of a data set, while neglecting to validate the ground truth incurs various negative consequences when used in anomaly detection scenarios. For example, inaccurate labelling of samples within a data set causes a distorted view of the normal behaviour profile, which further affects the accuracy of any decision making processes taken by ML algorithms accordingly.

The time, effort, and expertise required to perform manual analysis or implement automatic labelling and validation mechanisms has been suggested to be the cause of such a challenge within the research community. The data sets identified in this thesis attempt to provide a ground truth through various methods. However, each of these approaches incur limitations in some manner. For example, one of the most recent intrusion detection data sets, CICIDS2017, provides a ground truth as the attacks are known before-hand and injected into simulated benign traffic. There are a number of problems with this approach. As the modern threat landscape is constantly evolving, it should be noted that the cherry-picked primitive attacks performed multiple years ago are no longer representative of current threats. Furthermore, the normal behaviour profile is synthetically generated, which indicates that there exists differences between it and benign activity captured through the deployment of network services in a real-world network environment. When leveraging these data sets to provide intrusion detection capabilities, it is essential that the normal behaviour is as realistic and appropriate to the environment as possible to provide accurate results.

Correctly labelling the data set ensures all attacks are successfully identified, separating the benign traces from the malicious traces, thus revealing the true nature of the data. This is useful in many scenarios. For example, when taking a supervised ML approach, these systems require labelled data sets to be trained. Moreover, in more general scenarios the real nature of a data set must be known in order to evaluate the effectiveness of the detection approach, i.e. to determine the True Positive (TP) and False Positive (FP) rates of detection. Another important scenario is the feature selection process, the analysis and identification of feature importance within a data set, which can only be utilised if labels are included. Within typical network conditions, collecting labelled data sets is impossible [115].

At present, the majority of data sets are labelled manually by a technician performing forensic analysis. This is impractical especially when the amount of data is large, taking considerable amounts of time, thus, not enabling online implementation. As a further compromise, attacks which are deemed representative of the type the IDS should detect are sometimes *injected* into the data set. As discussed in the previous chapter, the data sets based on this method face challenges such as blending the attacks into the normal traffic in a realistic manner. As a result, some data sets have inherent problems, which limit the evaluation of IDSs. Furthermore, none of the data sets identified in this thesis which inject attacks, have provided details about why they chose the attacks they did, and how they were identified as being representative of attacks currently propagating. As common IDS approaches, such as classification through ML, find similarities between the known attacks and future attacks [22], it is essential that these attacks be relevant for the IDS to provide an effective response.

Another approach is detailed in Sommer's seminal work on ML for NIDS outside the closed world, exploring how the ground truth of a data set can be gathered via a mechanism orthogonal to how the detector works [22]. This approach has gathered traction within the research community, with frameworks emerging which are designed to automatically label data sets using methods unrelated to typical IDS detection methodology [115, 73, 113, 114]. This avenue enables a constant supply of emerging labelled data without the need for human intervention, effectively providing an autonomous evolving understanding of malicious behaviour to IDSs. Furthermore, the constant stream of labelled data has been suggested to stop the overstudy of data sets, or publication of irrelevant results on outdated data sets, helping alleviate some of the current problems facing the research community [23]. Research Question 2 poses the question of whether it is possible to establish a ground truth through correlation with CTI services. The approach adopted in this thesis is similar to the research discussed above, but it is the first to identify malicious behaviour through the use of diverse Internet vantage points.

- **Research Challenge 3:** Traditional anomaly detection methods are not suitable for the volume of data produced in modern networks

In Section 2.3, IDSs are described alongside their techniques and properties. These type of systems aim to detect attacks through various methods. The most common approach is the use of signature-based methods, which compare each suspicious event to a known database of malicious indicators of compromise. However, as previously discussed, this type of approach has the ability to detect only a small subset of attacks, and typically cannot detect unknown attacks, such as emerging threats and zero-days [130]. Therefore, in this thesis a machine learning approach to anomaly detection is taken, utilising classification algorithms to identify emerging threats of a similar ilk to what has been discovered from the labelled telemetry collected from honeypots. The data set output by Citrus is additionally designed with the intention to be used with machine learning based intrusion detection systems. The utilisation of machine learning to perform intrusion detection has been evaluated in modern literature, and has provided largely positive results through high detection accuracy.

Despite this, due to the explosive growth in the complexity of modern networks and propagation of malware, the cost associated with the processing of the corresponding training and test data used in the creation and evaluation of machine learning models is substantial. As a result, preliminary investigations have revealed that existing approaches to anomaly detection are not effective enough, especially upon the consideration of real-time or near real-time prediction [5].

Research Question 3 poses the question of whether big data technologies can deal with attack detection in large-scale networks. This approach would alleviate the growing scalability challenges associated with such varied and large amounts of data in modern networks [131, 132, 133, 134]. These emerging technologies provide a scalable solution for the storage, access, and processing of large amounts of heterogeneous data samples. Motivated by such contributions, the design of Citrus should integrate Big Data technologies to leverage their high performance data pre-processing capabilities in addition to their capacity to perform distributed computing and processing of both training and testing of machine learning models for the purpose of intrusion detection. Such methodology is suggested to help alleviate the relatively high computational cost of machine learning algorithms when compared to traditional signature-based methods.

While Big Data technologies help alleviate certain challenges incurred by the large volumes of data generated in modern networks, intrusion detection frameworks must operate in an online fashion to provide network administrators timely alerts for mitigation and remediation purposes. In modern literature, a great number of approaches which lack near real-time implementation have leveraged Big Data technologies to perform batch processing and offline classification of network telemetry [135, 136, 132, 137, 138, 97, 133]. Whilst providing a scalable intrusion detection solution, these approaches fail to provide timely remediation since the detection process occurs much later than the actual malicious behaviour. Additionally, further stages in the attack are allowed to continue, such as data exfiltration, which could be stopped upon the rapid identification of initial stages. These types of publications typically attempt to marginally increase detection rates on dated data sets, which, as identified in [23], is a current problem facing the research community. As a result, many research contributions fail to address the challenges faced when running classification algorithms based upon big data streaming [109].

A practical approach to intrusion detection, which is also suitable for the diversity and verbosity of data within modern networks, must implement a scalable near real-time decision making mechanism using streaming technology to ensure malicious behaviour is detected in a timely manner. To achieve this type of intrusion detection, telemetry must be transmitted to the intrusion detection framework where live classification occurs to determine whether malicious actions have taken place. In order to detect attacks currently emanating from devices from within the network, sensors must additionally be deployed to capture and transmit this telemetry, which contains the intricate properties of the infiltration attempt. Based upon the previously encountered behaviour in training data, a machine learning model can then make an informed decision about the nature of this new behaviour, and potentially alert administrators if it does not correspond to the normal profile.

The challenges outlined above showcase a variety of deficiencies within the literature. This thesis aims to provide contributions within these areas to alleviate these challenges. In order to achieve this, certain design requirements are defined which are motivated by these challenges. These design requirements are detailed in the following chapter.

2.5 Summary

This chapter has presented background and related work spanning various topics of research. Initially, the modern threat landscape is discussed, highlighting how threats are constantly evolving in order to propagate a continuous barrage of attacks against networks. This is further detailed through the examination of emerging botnet, malware, and threat actor methodology.

This chapter additionally discussed the need to monitor these emerging threats in order to mitigate infections and service disruption. Notably, the use of CTI to gain knowledge of cutting edge adversarial tactics and techniques has fostered much interest within the research community. The technologies enabling the creation of such CTI data have been highlighted within this chapter. This is initially outlined through a detailed review of honeypot mechanisms, which enable the gathering of rich malicious behaviour and corresponding compromise indicators. Third party threat intelligence services provide an additional source of malicious behaviour, further enhancing contextual understanding of suspect network events.

Bolstering intrusion detection mechanisms with such intelligence enables accurate detection of emerging threats. This is further explored in this chapter through an investigation into modern IDSs. IDSs are used to defend against threats which plague modern networks. They can be classified into either signature-based or anomaly-based systems. Signature-based systems use a database of known attack properties, while anomaly-based based systems require a profile which is representative of normal behaviour. Data sets which contain a normal behaviour profile are scarce and rarely publicly available. The data sets that are available have significant limitations which inhibit their practical use in IDSs. As discussed, CTI enables updates to the signature database of an IDS, preventing future compromise through the identified attack vector. Further benefits become evident when employed alongside an IDS which incorporates machine learning techniques.

A lot of effort has been made by the research community into the evaluation of machine learning based anomaly detection techniques used in the detection of network attacks. However, many approaches focus solely on the evaluation of offline, batch prediction. Such approaches neglect to examine streaming based anomaly detection, as used in time-critical, real world attack detection implementations, and its associated challenges.

The telemetry derived from CTI sources, such as honeypots, can be used to train machine learning models, which have the ability to prevent emerging attacks of a similar nature to what has been observed. As not all telemetry associated with honeypots is inherently malicious in nature, a robust ground truth is required to rigorously separate the benign traces. The development of a ground truth can be implemented in multiple ways. However, recent developments within CTI enable the correlation of suspicious behaviour for this purpose.

Finally, this chapter concludes with the identification of various challenges facing the research community. These challenges range from the quality of intrusion detection data sets to the practical application of such in realistic environments. In the next chapter, the design of Citrus is outlined, which attempts to address these challenges.

Chapter 3

Design of Citrus

In this chapter, the design of a modern intrusion detection framework, Citrus, which has the capacity to detect emerging threats is outlined. This is initially achieved through the discussion of Citrus' design requirements in Section 3.1. These requirements document the core functionality that the proposed system should incorporate based upon the challenges identified in the previous chapter. Subsequently, Section 3.2 presents the architecture and design overview of Citrus. The security considerations of the Citrus framework are divulged in Section 3.3, and the privacy requirements of Citrus are discussed in Section 4.4.

3.1 Requirements

This section highlights the high level design requirements and core functionality desired for a modern intrusion detection framework, which aims to tackle emerging threats.

3.1.1 Attack Data Availability

As evidenced by the background and related work in Chapter 2, the modern threat landscape incorporates a myriad of evolving attacks, which have the ability to cause devastating damage to network infrastructure. Research Challenge 1, as mentioned in Section 2.4, states that the security community lacks high-quality modern data sets which reflect these evolving attack vectors. These data sets are essential for the training of machine learning models and the evaluation of intrusion detection mechanisms. Through a comprehensive literature review, honeypots have been identified as a means to attract cyber threats and capture relevant emerging attack properties.

Motivated by this challenge within research, an outline of several design requirements are provided. These requirements are intended to deliver contributions into the cyber security field through the composition and public release of an innovative intrusion detection data set. These are as follows:

1. Leverage honeypots to continuously capture and collect network telemetry incorporating real emerging attacks
2. Engineer features from the captured telemetry to compile an updated intrusion detection data set

As discovered through literature review, a substantial number of current intrusion detection data sets inject attacks to incorporate malicious behaviour. This approach often creates limitations in the form of realism as attacks chosen by the authors must be blended seamlessly with the traffic representative of normal behaviour. In addition, a new data set must be authored to reflect changes in attack patterns. The deployment of honeypots to compile an intrusion detection data set ensures up-to-date attacks, which are representative of real attacks currently being orchestrated, are captured. Furthermore, this approach enables the constant capture of novel and emerging attacks which are part of the perpetually evolving cyber threat landscape.

The attacks captured by these honeypots are used to compile a new intrusion detection data set, which receives periodic updates. The purpose of the data set is to evaluate attack detection mechanisms, specifically using machine learning based approaches. The design of Citrus should incorporate this functionality, collating telemetry from various honeypots, and pre-processing it in preparation for the labelling stage. Ultimately, the data set should be released to the general public to promote further contributions into the intrusion detection domain.

3.1.2 Ground Truth Development

When taking a supervised machine learning approach to the detection of emerging threats, a labelled data set containing benign and attack telemetry representative of the modern threat landscape and of a recent nature is required. In order to label a data set, a ground truth is required. As identified by Research Challenge 2 in Section 2.4, the number of publicly available data sets which establish and validate the ground truth are limited, and pose additional limitations, which suggest that they are not suitable for this purpose. The following high level design requirements attempt to help alleviate this challenge:

3. Automatically develop a ground truth for captured network telemetry using contextual CTI

Through the capture of telemetry from honeypots, a robust data set will be made available, which details emerging attack properties. Furthermore, within the telemetry exists benign communications to internal services such as databases used to transfer and securely store the logs, as well as scanning from external services, which aim to profile the Internet topology for legitimate purposes, and traffic which is a side effect of an attack but cannot be classified as malicious by itself [104]. In order to successfully implement an automated labelling method to separate the benign from malicious traces, a robust ground truth must be developed. This approach is explored in the design of Citrus through the realisation of these requirements.

Citrus is the first framework to correlate behaviour observed locally with third party CTI services in order to establish a ground truth for the purposes of bolstering AD algorithms. This enables Citrus to gain a fuller understanding of how actors conduct their behaviour with regards to the general Internet, and provide associated labels for the published data set. These actors are identified through the extraction of IP addresses from telemetry captured by honeypots. As previously discussed, the majority of CTI services deploy bespoke servers at particular locations to gain a vantage point of malicious communications or behaviour. They typically expose an API, which allows clients to interface with the system and collect information relating to their deployments.

From the additional data obtained through these sources, intricate details pertaining to infiltration attempts observed elsewhere and other malicious behaviour are revealed. This intelligence enables an insight into the relationships between every IP address encountered by deployed honeypots and various entities. Due to the heterogeneity of CTI services, a plethora of different relationships can be identified, such as being active within blocklists, communicating with a given malware sample, belonging to a particular AS, and exposing certain services.

Leveraging concepts of graph theory, these relationships can be modelled in a visually intuitive representation. The ground truth is further developed from this representation, and the design of Citrus should incorporate this functionality in order to provide a novel automatic labelling mechanism for the data set.

3.1.3 Near Real-time Attack Detection

An intrusion detection framework is typically used in order to detect attacks which are manifested within the network. There exists a number of different approaches to intrusion detection. However, machine learning based anomaly detection techniques have recently demonstrated excellent accuracy under evaluation scenarios. Systems which employ these techniques require training data consisting of normal behaviour and attack patterns to build machine learning models. In order for the prompt detection of attacks, frameworks must be instrumented in a such a manner which enables iterative prediction on live data using these models.

As identified in Research Challenge 3, traditional anomaly detection methods are rendered ineffective due to the vast amount of highly-dimensional heterogeneous data emanating from devices within modern networks. In order to satisfy the data processing requirements for near real-time detection, innovation within intrusion detection systems is required. Recently, technologies have emerged which are designed to alleviate challenges related to the processing of Big Data. These technologies typically leverage a cluster of compute resources to perform efficient processing in parallel. This is enabled through the segmentation of data amongst the cluster. In order to bolster anomaly detection mechanisms with the ability to successfully classify and identify attacks in large amounts of data within an acceptable time frame, the following design requirement of Citrus is established:

4. Leverage system and data parallelism to perform near real-time detection of emerging threats

To achieve this functionality, Citrus requires a stream of telemetry emanating from within the local network. Upon receipt of the telemetry, Citrus should provide a robust data pipeline, which distributes it amongst nodes in a cluster, in preparation for parallel processing. Furthermore, Citrus should be instrumented to support the online classification of traffic using machine learning models, which have been previously trained offline using data representative of local network conditions. This online classification provides a means to rapidly identify malicious behaviour and alert network administrators of potential misuse, thus, acting as a viable approach for next generation network defense solutions.

3.2 Citrus System Architecture

Citrus is a framework which provides automated network telemetry labeling, and scalable intrusion detection mechanisms. In this section, an overview of Citrus' architecture is presented, detailing the new components created and technologies utilised to achieve the aforementioned requirements.

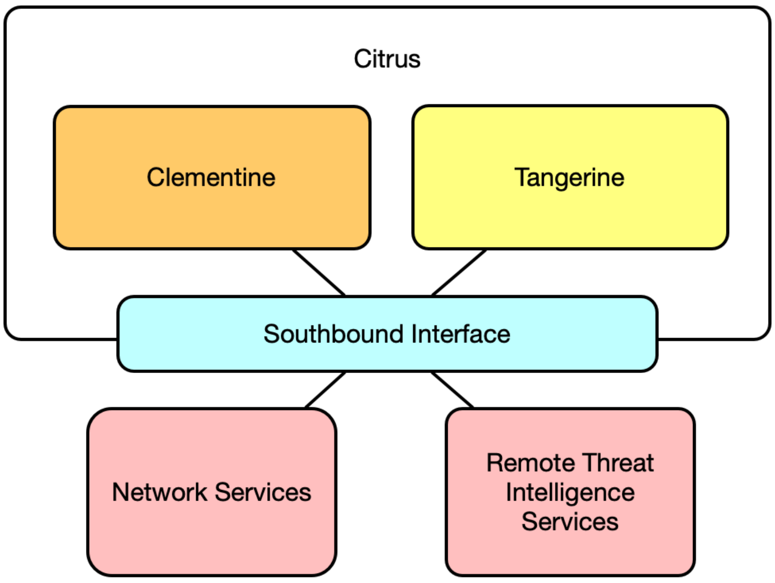


Figure 3.1: Citrus architecture design

The architecture of Citrus, as illustrated in Figure 3.1, is composed of distinct components which interface with services deployed within the network, as well as remote services located on the Internet. The southernmost components within Figure 3.1 represent these services, which provide Citrus crucial input data necessary for its operation. Furthermore, they are also utilised for output operations, such as saving labelled telemetry to disk for future dissemination within the research community.

These services interact with Citrus through its Southbound Interface, which in turn feeds the request into to the appropriate higher level component: Tangerine or Clementine. This Southbound Interface, which is further discussed in Section 3.2.1, provides Citrus the ability to integrate with Big Data technologies, enabling an exceptional level of performance through their distributed storage, access, and processing capabilities. Moreover, the Southbound Interface additionally facilitates the communication between Citrus and remote threat intelligence services, providing rich contextual information for suspect historic telemetry captured by various honeypots.

Above the Southbound Interface is where both Tangerine and Clementine are located. These two components are independent of each other and offer distinct functionality. The Tangerine component, which is further outlined in Section 3.2.2, is responsible for the collection of historical telemetry. This is composed of both attack telemetry from honeypots and benign telemetry from internal services. Tangerine also orchestrates the collection of corresponding contextual CTI. Through this process, Tangerine is also able to provide target labels for the historical telemetry based upon the derived contextual information and development of a ground truth, enabling a constant supply of labelled flow-based telemetry suitable for the evaluation of Intrusion Detection Systems.

Clementine exists at the same level as Tangerine; receiving input data from services deployed within the network, which traverse through the Southbound Interface in the process. This component, which is described in more detail in Section 3.2.3, orchestrates intrusion detection procedures of online flows emanating from devices within the network. Critically, Clementine utilises the flow-based labelled data set output by Tangerine to achieve this. Due to the large amount of high-dimensional telemetry captured within the network, a scalable solution is required to provide an efficient response. Clementine achieves this through integration with emerging Big Data technologies, namely, high-throughput distributed streaming frameworks to transmit telemetry in an efficient fashion, and distributed processing frameworks to execute various tasks in parallel.

3.2.1 Southbound Interface

The Southbound Interface is composed of various modules that are used to communicate with services located within the network and on the Internet. These modules provide Citrus rich input data used for both automated labelling and online intrusion detection purposes. They additionally enable Citrus to perform output operations, further aiding its ability to integrate with emerging technologies. This section details each module located within the Southbound Interface, providing an overview of its operations and responsibilities.

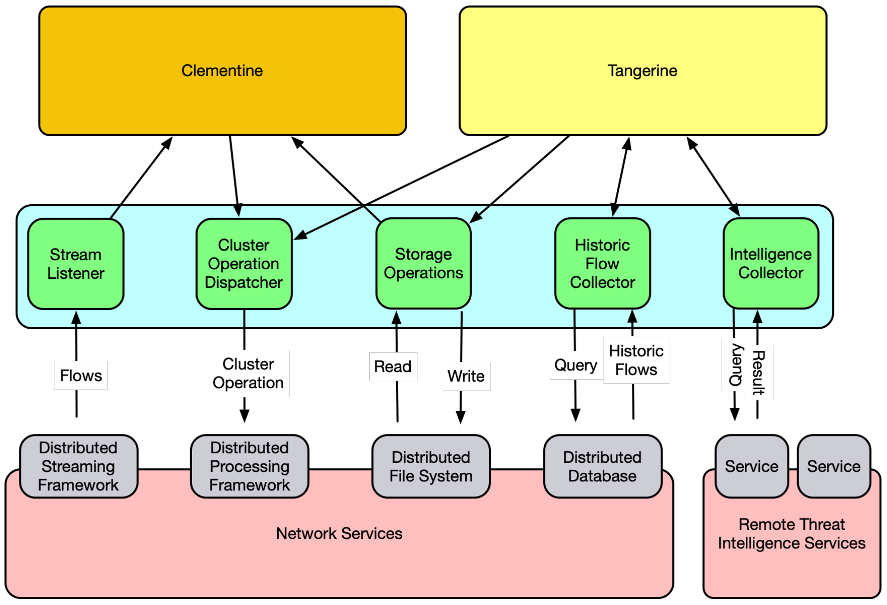


Figure 3.2: Southbound interface design overview

3.2.1.1 Stream Listener

The first module within the Southbound Interface is the *Stream Listener*. This module solely serves as an input to Clementine. Clementine uses this input to perform online flow-based intrusion detection. As such, the information flowing through the *Stream Listener* consists of network flows emanating from devices located within the network. Network flows are transmitted to the *Stream Listener* using high-throughput streaming technologies, enabling rapid access to suspicious flows for further intrusion detection purposes.

Due to the way in which this module is designed, a plethora of streaming technology can be used. However, this research leverages Apache Kafka [98] due to it being able to efficiently handle high amounts of load in a distributed fashion. Kafka is integrated with this module through programmatically provisioning Kafka consumers, which receive and process data from the message broker. This data is formatted depending on the manner of capture. The format chosen within this research is a JavaScript Object Notation (JSON) formatted string.

3.2.1.2 Cluster Operation Dispatcher

The next module is the *Cluster Operation Dispatcher*, which provides an interface between Citrus and distributed processing frameworks. This module utilises clusters of executor nodes deployed within the network to perform parallel computation of complex operations. This integration enables highly efficient transformations of vast, high-dimensional intrusion detection data sets. Clementine additionally utilises this interface to submit machine learning tasks to executor nodes, significantly decreasing the time taken for training and prediction of large data sets when compared to traditional approaches. This is achieved through integration with a library provided by the distributed processing framework. This library exposes many methods to manipulate data, which is partitioned across many physical nodes.

3.2.1.3 Storage Operations

The *Storage Operations* module is responsible for the reading and writing of labelled data sets and trained models to a distributed file system. This functionality is critical for a a variety of use cases. Namely, this enables Tangerine the ability to write labelled data sets to disk. As a result, this also allows Clementine to access labelled telemetry in a distributed fashion, enabling rapid access and decreasing loading time as records are partitioned between executor nodes. Furthermore, this also allows models to be saved for future use, providing re-usability for objects, which incur prolonged training times. The *Storage Operations* module also enables the resulting labelled flow-based data set to be shared with the wider research community, with the intention of providing daily submissions of realistic, emerging attack telemetry which next generation intrusion detection systems can use to evaluate their approach.

3.2.1.4 Historic Flow Collector

The *Historic Flow Collector* module provides an interface between Tangerine and a distributed database. The distributed database stores historic flow-based telemetry emanating from honeypots, in which attack traffic occurs, and services deployed privately within the network, in which benign traffic occurs. The module allows Tangerine to query a database for telemetry using an expressive language, which returns each record matched by the query. This is achieved through sending web requests to the database and receiving back data in JSON format.

Critically, this module enables Tangerine to load vast amounts of telemetry into memory, which will then be used to compile a robust intrusion detection data set. Therefore, this module is intended to satisfy the Design Requirement 1, further outlined in Section 3.1.1, as it collects traffic captured by diverse honeypot deployments.

3.2.1.5 Intelligence Collector

Unlike the other modules discussed in this section, the *Intelligence Collector* module provides an interface between Citrus and remote services located on the Internet. Through this module, Tangerine is able to correlate the data derived from honeypot deployments with various CTI services to gain contextual information regarding suspect attackers. The CTI services leveraged by Tangerine are heterogeneous in nature and provide varying information. Most notably, intelligence services which provide access to various blocklists will be communicated with to corroborate whether a host which interacts with the honeypots Citrus governs is known to be an attacker, confirming that the host responsible is acting in a overtly malicious manner. There is also further integration with auxiliary threat intelligence services, such as malware analysis platforms, which provide data that details malware samples that are associated with a suspect attacker. As illustrated within Figure 3.2, the *Intelligence Collector* module provides both input and output operations. This is to enable custom queries (as an output) to each distinct threat intelligence service, such as providing a list of suspicious hosts, and returning to Tangerine (as an input) the corresponding response from the service.

3.2.2 Tangerine

The *Tangerine* component within Citrus, as illustrated in Figure 3.3, performs automated intrusion detection data set labelling through correlation with CTI service providers. This component ultimately outputs a comprehensive flow-based labelled data set, which enables the *Clementine* module to perform online

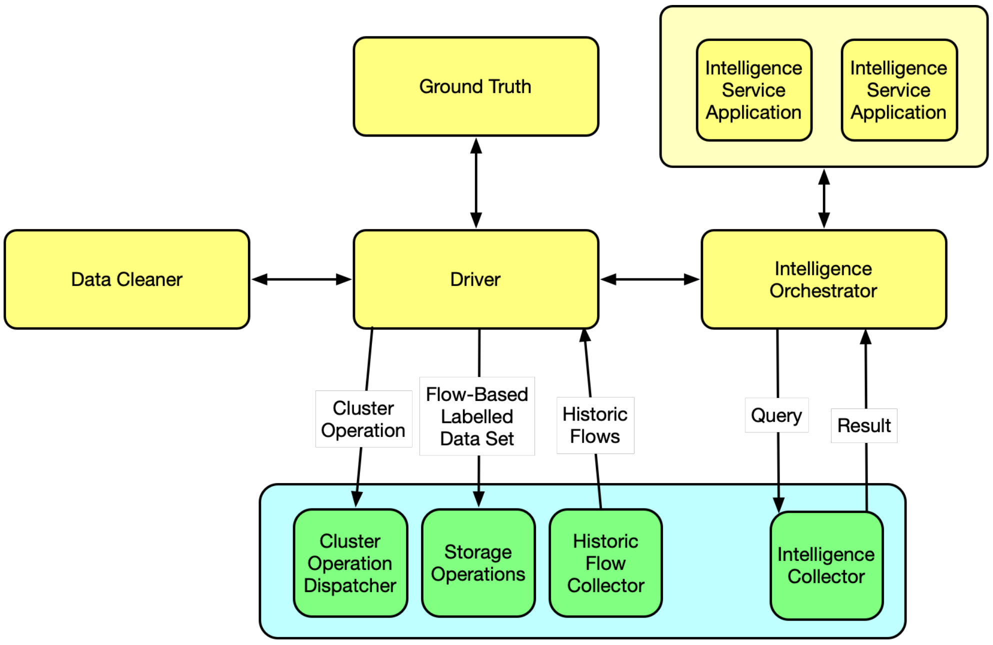

Ground Truth
Intelligence Service Application
Intelligence Service Application
Data Cleaner
Driver
Intelligence Orchestrator
Cluster Operation
Flow-Based Labelled Data Set
Historic Flows
Query
Result
Cluster Operation Dispatcher
Storage Operations
Historic Flow Collector
Intelligence Collector

Figure 3.3: Tangerine design overview

intrusion detection tasks.

This module is designed to develop a ground truth for telemetry captured on a specific day. Therefore, this module should begin execution upon the discovery of new telemetry captured over a period of 24 hours. As a result, a continuous feed of labelled telemetry is supplied by *Tangerine* through ground truth validation.

To achieve this functionality, *Tangerine* initially gathers the telemetry to be labelled from the Southbound Interface. The telemetry is then pre-processed in preparation for exportation as a labelled data set. Contextual information is then gathered through correlation with CTI services. Upon successful collection and processing of this contextual information, the relationships between potential attackers observed within the telemetry on the capture date are extracted. This is achieved through integration with a graph library. A clustering technique is then applied to the nodes, i.e. the suspected attackers, within the graph to identify highly connected *supernodes*. These *supernodes* are determined to be malicious as they are associated with a large number of blocklists or reports derived from CTI services. Based upon this clustering approach, the malicious communications within the collected telemetry are able to be accurately labelled.

These stages within the ground truth development process are embodied as components within Tangerine. The remainder of this subsection details these components and discusses how they work in harmony to produce a continuous supply of flow-based labelled telemetry data.

3.2.2.1 Driver

The *Driver* component is the entry point of Tangerine, and is additionally responsible for the orchestration of all stages within the labelling process. As a result, this component has connections with all of the other components within Tangerine. The *Driver* has one input and produces two distinct output operations. The input is in the form of flow based telemetry data collected from honeypots and benign servers deployed within the network, which is delivered through the Southbound Interface to the *Driver* component.

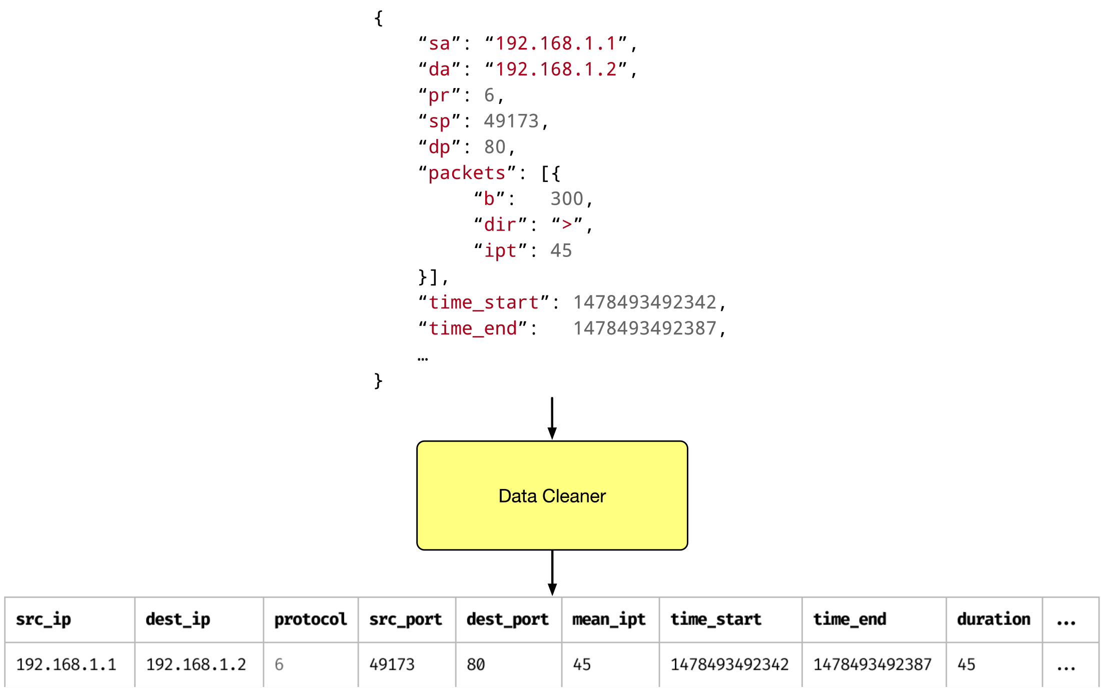
Due to the large amount of telemetry data needed to be stored in memory and operated upon, Tangerine integrates with Big Data processing frameworks to alleviate scalability challenges. The *Driver* component triggers operations to be performed on a cluster of worker nodes, which transforms the telemetry in parallel for data preparation purposes. Upon successful response from all components within Tangerine, which indicates that the telemetry data has been cleaned, contextualised, and labelled, the *Driver* is able to produce a flow-based labelled intrusion detection data set as an output destined for storage within a distributed file system.

3.2.2.2 Data Cleaner

The *Data Cleaner* component is responsible for the transformation of telemetry data into an appropriate format. All telemetry data gathered from honeypots and benign servers deployed within the network is processed through this component to prepare the relevant features required for an intrusion detection data set. As a result, the *Data Cleaner* plays a critical role in delivering the telemetry data in a format appropriate for the evaluation of Intrusion Detection Systems. Furthermore, the design of this module satisfies Design Requirement 2 as its purpose is to engineer features relating to diverse traffic measurements.

This component receives from the Driver raw telemetry data in the form of network flow measurements. The *Data Cleaner* transforms the raw telemetry data and returns it to the Driver in a standard format, preparing it to be output as a data set. As illustrated in Figure 3.4, the telemetry data is fed into the *Data*

Cleaner in the form of JSON formatted string. There are many more features, but they have been omitted for brevity. For example, the additional features include both flow-based and packet-based features, such as bytes_in and entropy. Further information regarding all of the features can be found in Section 5.2.5. The *Data Cleaner* then constructs additional features and converts the data into a columnar format. The columnar format is required by the cluster processing framework to perform distributed operations.



```
{
    “sa”: “192.168.1.1”,
    “da”: “192.168.1.2”,
    “pr”: 6,
    “sp”: 49173,
    “dp”: 80,
    “packets”: [{
        “b”:   300,
        “dir”: “>”,
        “ipt”: 45
    }],
    “time_start”: 1478493492342,
    “time_end”:   1478493492387,
    …
}
```

src_ip	dest_ip	protocol	src_port	dest_port	mean_ipt	time_start	time_end	duration	...
192.168.1.1	192.168.1.2	6	49173	80	45	1478493492342	1478493492387	45	...

Figure 3.4: Data transformation

3.2.2.3 Intelligence Orchestrator

The *Intelligence Orchestrator* component coordinates requests for CTI data. It achieves this through the instantiation and governing of *Intelligence Service Applications*, custom applications built specifically for communication with a certain CTI service. As such, this component plays a critical role in enabling extensibility and flexibility for future integrations with new CTI services. Critically, through the orchestration of these *Intelligence Service Applications*, this module helps to develop a ground truth.

The *Intelligence Orchestrator* receives a list of IP addresses, which are extracted from the telemetry data collected by the Driver. These IP addresses uniquely identify potential attackers interacting with the honeypots deployed within the network, and are utilised to perform queries on CTI services to gain contextual information relating to these actors. The *Intelligence Orchestrator* instructs each *Intelligence Service Application* to construct a list of request objects, using the IP addresses as a parameter within the request, to query the corresponding CTI service. These request objects are transferred to the Intelligence Collector module within the Southbound Interface, where each request is processed and the corresponding response from the service is returned.

Upon successful response from the Southbound Interface, the *Intelligence Orchestrator* then instructs each *Intelligence Service Application* to parse the response. When all responses have been parsed, the resulting CTI for each actor observed within the telemetry data is then transmitted to the Driver.

3.2.2.4 Intelligence Service Application

Intelligence Service Applications are developed within Tangerine, which contain the functionality required to query and parse responses from threat intelligence services. As these services are heterogeneous in nature, there is no standard method in which to construct a query and parse the corresponding response. As a result, to enable communication between Tangerine and specific CTI services, a corresponding application must be instrumented, which contains this distinct functionality. These applications provide their procedures to the Intelligence Orchestrator, which further coordinates the intelligence gathering process.

3.2.2.5 Ground Truth

The *Ground Truth* component is responsible for the development of a ground truth relating to telemetry data collected by Citrus. The functionality provided by this module is intended to satisfy Design Requirement 3, which is discussed in Section 3.1.2. The approach taken identifies supernodes within a graphical representation of suspected attackers and their relationships as discovered through active CTI correlation. As mentioned previously, services which provide access to CTI data enable an understanding of malicious behavioural patterns observed on various parts of the Internet. As a result of the integration with CTI services, this module is designed to create and manipulate graph data structures based upon the contextual information received from CTI services regarding potential attackers. In detail, the *Ground Truth* module contains the functionality required to add nodes and connections to others through edges. These connections are intended to indicate a relationship between nodes, such as being active within a blocklist.

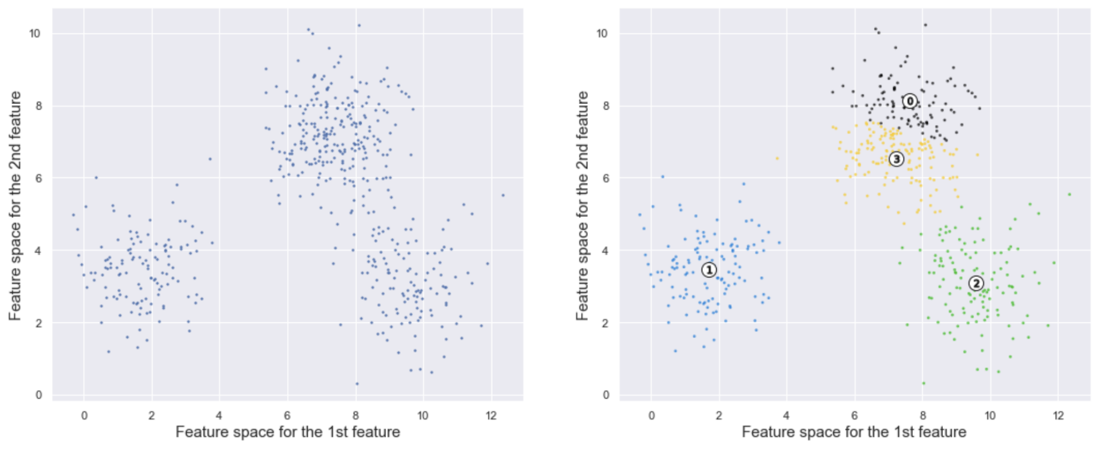


Figure 3.5: Graph feature clustering

In addition to the graphing capabilities, the *Ground Truth* module is designed to perform clustering of features extracted from the graph. Upon completion of graph related processing, these features are input into a clustering algorithm, which separates the nodes into clusters based upon the features. The clusters are then used to identify supernodes, which are the nodes established as having many relationships with malicious entities as derived through CTI correlation.

This approach is illustrated in Figure 3.5. The graphs within the figure are populated with data points extracted from features of the graphical representation of attackers and their associations as derived through correlation with CTI services. As shown, they can be visualised in two-dimensional space. The left graph shows the data points before clustering, while the graph on the right displays the result of clustering. The cluster centres are also clearly marked to illustrate the data points which belong to a particular cluster.

This determination ultimately provides the automated labelling methodology inherent within Tangerine. Therefore, based upon this clustering approach to identify supernodes, the telemetry data associated with each node within the graph can be labelled. Upon this successful identification of supernodes, the *Ground Truth* component provides a list of labels to the Driver. The Driver then performs transformation on the telemetry data to include such labels, and prepares it for output as a labelled data set.

3.2.3 Clementine

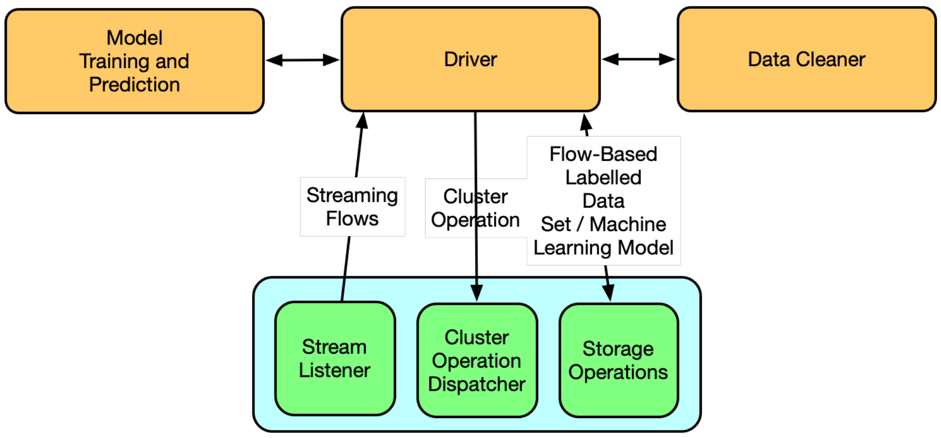

Model Training and Prediction
Driver
Data Cleaner
Streaming Flows
Cluster Operation
Flow-Based Labelled Data Set / Machine Learning Model
Stream Listener
Cluster Operation Dispatcher
Storage Operations

Figure 3.6: Clementine design overview

The *Clementine* component within Citrus, as illustrated in Figure 3.6, performs online intrusion detection of flows emanating from devices within the network. These network flows are transmitted to *Clementine* through the Southbound Interface using emerging streaming technology. This component adopts a machine learning approach to intrusion detection. In this approach, supervised classification models are trained on the labelled flow-based data set output by Tangerine. Both the initial training data set and the streaming flows are input to Clementine through the Southbound Interface. Both of these are considerable in size and dimensionality. As motivated by contributions within the research community, *Clementine* integrates with Big Data technologies to execute training and prediction tasks in parallel across distributed worker nodes. In addition, this component also performs the transformations required for the pre-processing of streaming flows in preparation for prediction. This functionality is divided into distinct components within *Clementine*. The remainder of this section details these components and their role in providing online intrusion detection capabilities.

3.2.3.1 Driver

The *Driver* within Clementine fundamentally coordinates all stages within the intrusion detection process. Initially, this component receives, as an input, a flow-based labelled data set from the Southbound Interface. The *Driver* then instructs the Model Training and Prediction component to build a trained classification model from this data set. To achieve this in an efficient fashion, training operations are dispatched to a cluster of executor nodes, which process the request in parallel. Upon the successful training of the model, Clementine is ready to begin predicting whether streaming flows emanating from devices within the network are of a normal or malicious nature.

The *Driver* then begins to consume network flows transmitted through the Southbound Interface. These flows are pre-processed through the Data Cleaner to prepare the relevant features used for flow classification. The *Driver* then instructs the Model Training and Prediction component to predict the label associated with the flow.

3.2.3.2 Data Cleaner

The *Data Cleaner* is responsible for the pre-processing of streaming network flows. This component periodically receives raw network flow telemetry data from the Driver. The *Data Cleaner* parses each flow and creates the appropriate features used in the prediction process. Upon successful creation of features, they are then returned to the Driver.

3.2.3.3 Model Training and Prediction

The *Model Training and Prediction* component provides the machine learning capabilities used by Clementine to perform intrusion detection. To achieve this in an efficient fashion, Clementine integrates with high-throughput streaming frameworks and Big Data processing frameworks, which provide highly scalable machine learning algorithms. Critically, these frameworks enable this module to leverage data and system parallelism for use in intrusion detection tasks, which is intended to satisfy Design Requirement 4.

This component performs two critical tasks in the intrusion detection process: the training of a machine learning model on a flow-based data set containing both benign and malicious traces, and the prediction of streaming flows emanating from devices within the network. Both of these tasks require features of flows to be sent from the Driver. Upon the receipt of these features, training and prediction operations are executed on a cluster of worker nodes. In the event of training operations, the trained model is returned to the Driver. In the event of prediction tasks, the associated predicted label for each flow is returned to the Driver, providing effective detection of emerging threats emanating from devices within the network.

3.3 Security Considerations

Considerations must be made for the security of Citrus in order to provide a robust framework. Citrus was designed with this in mind and many choices were made to architect a secure solution. One example of such a consideration is that Citrus does not need to be directly accessible from the Internet to function. All communication with external networks is initiated by Citrus through Hypertext Transfer Protocol Secure (HTTPS) to ensure confidentiality and integrity of data transmission. This means that Citrus can be placed behind a Network Address Translation (NAT) gateway. In addition, Citrus does not expose any APIs to external sources, and instead provides an interface only for its own modules and internally managed services.

The managed services required by Citrus do not need to be accessible from the Internet. As a result, they can all be placed in private networks. This means that exploitation of these services, and Citrus, is very unlikely. The only caveat here is that communication must be allowed between honeypots and a database to store the telemetry data. Honeypots can be deployed with various placement strategies, including being accessible from the Internet. In this case, and more generally, it is recommended to follow the principle of least privilege. This ensures that applications and services can perform necessary functions but nothing more. For example, a network rule should be configured to enable the transfer of honeypot telemetry data to the database using a secure protocol. Any traffic that does not match this rule should be blocked. This will ensure that Citrus and the required services are totally isolated from any other infrastructure.

Despite the application and system security considerations, there exists potential avenues of exploitation through Citrus' use of ML algorithms. Since the internal workings of CTI services are kept secret, they should be considered as a black box. Citrus extracts data from these services to label training data. As a result, if these services tampered with the data, it could impact Citrus' labelling. For example, assigning normal behaviour to traces that are in fact malicious. This would also affect the ability of ML algorithms to predict unseen data based upon the tainted training data.

3.4 Privacy Requirements

The honeypot telemetry captured and used by Citrus incorporates a wide range of data. This includes features that could be considered Personally Identifiable Information (PII) such as IP addresses. Despite no communication being initiated by the honeypots, it is important to respect the privacy of individuals. Citrus removes such data and instead replaces it with more general information. In the case of IP addresses, they are converted to the number corresponding to the Autonomous System (AS) that the IP address belongs to. This has the benefit of removing personal information yet preserving some aspect of the data, which could still be useful.

3.5 Summary

This chapter has outlined an architecture which has been designed to achieve the aforementioned requirements detailed in Section 3.1. The design choices made within this chapter highlight the necessity for a scalable and efficient approach to intrusion detection through integrations with state of the art technologies. These choices are made to provide an efficient solution when dealing with extremely large amounts of data. Moreover, the extensibility of the system is explored through the design of Intelligence Service Applications, which can be used to communicate with novel threat intelligence platforms. These bespoke applications are designed to gather contextual information pertaining to potential attackers, and further enable the development of a robust ground, which separates malicious and suspicious traffic traces.

Citrus is designed to be deployable within a range of networks. The Southbound Interface ensures that integration with a variety of local network environments is seamless through the utilisation of streaming technologies to collect live telemetry data emanating from network devices. The design of Citrus is also intentionally agnostic to telemetry data it collects, labels, and uses to classify malicious events. As such, network devices can be instrumented to transmit packets or flows to Tangerine's distributed database for future labelling. This enables flexibility for the network operator to determine which is the most appropriate for their particular use case. In the following chapter, the design of Citrus is realised through a robust implementation.

Chapter 4

Implementation

In the previous section, the design of Citrus' architecture was presented. This architecture provides two overarching pieces of functionality, which enable the detection of emerging threats. These are embodied as components within Citrus: Tangerine and Clementine. The implementation of these core components is outlined in Section 4.1. Tangerine, which is further discussed in Section 4.1.3, orchestrates the automatic collection and labelling of telemetry incorporating emerging attacks. The intelligence service applications which enable correlation with CTI services are also outlined to demonstrate Citrus' intelligence gathering capabilities. Clementine, as discussed in more detail in Section 4.1.4, coordinates the consumption of streaming flows emanating from devices within the network, and utilises the labelled telemetry output by Tangerine to perform intrusion detection. As discussed in the design chapter, both of these components integrate with emerging technologies to perform their respective operations. The integration with these technologies is achieved through the implementation of a Southbound Interface, and is discussed in more detail in Section 4.1.1. The implementation of development and deployment tools are also outlined in this chapter. In Section 4.2, the implementation of a tool which enables the automatic publication of the compiled data set to GitHub [139] is discussed.

4.1 Citrus

This section discusses the components which compose the architecture of Citrus. In particular, the implementation detail of each component is divulged to highlight how the design architecture has been realised. Each of the components listen within this section have been implemented using the Python programming language [140]. Python version 3.7 was chosen as Python 2 has reached its end of life and is therefore no longer actively being maintained. Additionally, Python 3 introduces a range of new functionality to the language and also contains the largest number of libraries.

4.1.1 Shared Library

The components within Citrus share a range of common functionality. To reduce the amount of code duplicated within Citrus, a shared library is developed, which contains this common functionality[1]. The library is implemented as a Python package. The relevant components within Citrus have the ability to import this library to provide additional utilities and features.

As discussed in the design section, Citrus integrates with various Big Data technologies to perform efficient processing of vast high-dimensional data. Motivated by contributions identified in literature, this is implemented in Citrus through the utilisation of Apache Spark [13]. Apache Spark provides large scale data processing capabilites. It also supplies high-level APIs in Scala, Java, Python and R, which support structured data processing, machine learning and structured streaming for incremental computation and stream processing. This support also enables interoperability with other technologies which further aid large scale data processing, such as Elasticsearch [12], Apache Hadoop [10], and Apache Kafka [98]. The use of these technologies in discussed further in this chapter.

Apache Spark has the ability to execute processes on a cluster of executor nodes. This is coordinated by the SparkContext object within the application. SparkContext connects to various cluster managers (Standalone, Mesos [141], YARN [10], or Kubernetes [142]), which allocate resources across clusters. Upon allocation, Spark acquires executor nodes within the cluster, which perform computations and store data for an application. Finally, SparkContext sends *tasks* incorporated within an application to be run on the executors within the cluster in parallel.

Due to the development of Citrus being undertaken in the Python programming language, the Python bindings, PySpark [143], were used to successfully

[1]https://github.com/ruzzzzz/Citrus/tree/master/citrus_lib

integrate with Apache Spark. As this technology is utilised by both the Clementine and Tangerine components within Citrus, the shared library includes support for the execution of distributed tasks on executor nodes. This is achieved through instantiation of the SparkContext object, which is made available to other components within Citrus.

The telemetry used by Citrus to perform intrusion detection is initially collected in a non-standard format, and must be pre-processed for further operations. Tangerine must incorporate this functionality to prepare the telemetry for exportation as a labelled data set. Clementine also requires this functionality to prepare streaming telemetry emanating from devices within the network for classification. As a result, the shared library includes a *Data Cleaner* for this purpose. The Data Cleaner is implemented as an abstract class which supplies a *clean* method. This method is inherited by its child classes, which model various input sources, and pre-processes the data into a standard format. This enables extensibility for the collection of telemetry, which is created using various implementations. The implementation of the Data Cleaner enables the preparation of data and engineering of features, thus, it is intended to satisfy Design Requirement 2.

Citrus requires the configuration of several variables before it can begin execution. These include the IP address and port number of network services, and keys used in the authentication stage of communication with the threat intelligence service. For this purpose, Citrus contains a configuration file which can be retrieved by any module which requires it. The format of the configuration file is JSON. This format was chosen due to its self-descriptive nature, as well as library support within Python enabling the loading of such JSON strings into dictionaries within memory. A Python module is additionally implemented within Citrus' shared library to perform this functionality.

4.1.2 Southbound Interface

As discussed in Section 3.3.1, Citrus contains a Southbound Interface to integrate with various network services[2]. This interface is implemented as a Python module. Upon initialisation, this module instantiates each individual component within the Southbound Interface. Notably, the SparkContext object, obtained

[2]https://github.com/ruzzzzz/Citrus/tree/main/southbound

from the shared library, is passed to the relevant components constructor to perform distributed operations upon a cluster of executor nodes. The components contained within the Southbound Interface are accessed by the higher level components, Tangerine and Clementine, through a *get* method within the interface. This method accepts a string, which represents the component which is requested. The remainder of this section outlines each component within the Southbound Interface, and details the functionality each provide.

4.1.2.1 Stream Listener

The *Stream Listener* provides the functionality to consume streaming telemetry emanating from devices within the network. This enables the online evaluation of telemetry within a local network. This is achieved through integration with Apache Spark's streaming library [144]. Spark Streaming extends the traditional core Spark API by providing high-throughput, fault tolerant, and scalable stream processing of live data. This data can emanate from a variety of sources such as Apache Kafka [98], Kinesis [145], and rudimentary sockets. The streaming platform implemented within Citrus is Apache Kafka. This platform was chosen due to its distributed, replicated, and extremely highly performant nature. These attributes should help enable the timely online detection of emerging attacks.

The data streams can be operated upon using high-level functions such as map, reduce, and join. Furthermore, Spark Streaming additionally enables the application of machine learning algorithms upon such data streams. Spark provide a high-level abstraction of these streams of data, called Discretized Streams (DStreams). DStreams are internally represented as sequences of Resilient Distributed Datasets (RDDs). RDDs are a fault tolerant collection of elements partitioned across nodes within a cluster, which can be operated upon in parallel [146]. The sequencing of these elements structure a streaming computation as a series of micro-batch computations [147]. This property is leveraged by Citrus to perform online intrusion detection using conventional map and reduce operations as well as machine learning algorithms on batches with small time intervals.

To achieve this, the *Stream Listener* component provides three methods: *create*, *start*, and *stop*. Initially, the *create* method initialises a StreamingContext object, which is the entry point for all streaming functionality. This object is then passed, alongside the URI and port of the Kafka broker, to a function within

Spark, which creates an instance of a DStream object. This DStream object is returned to Clementine to perform parallel computations. The *start* method begins the consumption of telemetry from the Kafka broker, while *stop* terminates the streaming context.

4.1.2.2 Cluster Operations Dispatcher

RDDs are aware of the actions required to create themselves and can only be created from operations on either data in storage, or other RDDs [15]. These operations are named *transformations*, and include map, filter, reduce, join. The map transformation sends each element within the RDD through a function and returns the resulting RDD. The filter transformation discards elements within the RDD which do not satisfy a predicate. The reduce transformation performs aggregation of elements within the RDD. The join transformation combines elements within two distinct RDDs.

This module contains methods which perform these transformations upon network telemetry. An example of this is using the map transformation to pass each telemetry record through the Data Cleaner to pre-process it into a standard format. Furthermore, map transformations are utilised to construct additional features. These features augment existing data elements to produce features relevant in intrusion detection data sets. Such features which have been implemented include flow duration, mean inter packet arrival time, and target labels, and are detailed further in the following chapter.

4.1.2.3 Storage Operations

The *Storage Operations* module is responsible for read and write operations to distributed storage. Hadoop File System (HDFS) [10] was chosen as the storage medium due to its high-throughput access and fault tolerant nature. Upon initialisation by the Southbound Interface, this module's constructor is passed the SparkContext object as an argument. This object provides high level APIs, which include the ability to read and write items to HDFS. This is used within this module to implement operations for the loading and saving of labelled flow-based data sets and machine learning models.

This module provides a *read* and a *write* method to Citrus' high level components. To distinguish between whether a model or data set is being operated upon, a string is passed as an argument to each method. Based upon this argument and the name of the object, the corresponding location is deduced. Using the methods provided by SparkContext, the object is loaded/saved to the location within HDFS. In the case that a data set is to be loaded from storage, an RDD object is returned, which contains the data partitioned across nodes within a cluster.

4.1.2.4 Historic Flow Collector

The *Historic Flow Collector* module integrates with a distributed database containing raw network telemetry. This telemetry contains emerging attack data captured from honeypots, and benign communications between internal services. This enables the Tangerine component within Citrus to collect and label network telemetry for output as an intrusion detection data set. The database chosen to store this vast amount of data is Elasticsearch [12]. This is due to its distributed nature through sharding of data among nodes within a cluster. Each shard is an instance of a Lucene index, which enables the handling and querying of a sub-set of data stored in an Elasticsearch cluster.

Elasticsearch additionally enables the searching of large amounts of data in an efficient fashion. This module utilisies this functionality to query for a specific *type* of data stored within the database. This enables the flexibility to load telemetry of varying genres. Furthermore, Elasticsearch also separates data into indices. These indices can be considered a data organisation mechanism, which Citrus leverages to partition logs based upon the date of capture. This enables Citrus to seamlessly gather telemetry associated with a certain day which is required to be labelled.

This module provides a *query* method to Tangerine. Utilising the SparkContext object passed to the constructor, Elasticsearch is queried and returns each record that matches. These records are transformed by SparkContext to a RDD, which is then returned to the calling component. This module is implemented to collect honeypot telemetry data, and is intended to satisfy Design Requirement 1.

4.1.2.5 Intelligence Collector

The *Intelligence Collector* module is dissimilar to the other modules within the Southbound Interface, as it communicates with remote CTI services and does not require a SparkContext. The purpose of this module is to dispatch requests to these services and return the corresponding result. In order to perform a large number of HTTP requests in an efficient fashion, a multi-threaded library, grequests [148], was chosen to perform this task. This library combines the conventional requests [149] and gevent [150] libraries to perform HTTP requests concurrently using asynchronous design.

This module provides a *send* method, which accepts a list of request objects as an argument. These objects are individually composed of Intelligence Service Applications inherent within Tangerine. Each request is then dispatched using the grequests *map* function. Each response, including the response code and payload is appended to a dictionary. Upon all requests being dispatched and the corresponding response being received, this dictionary is then returned from the method.

4.1.3 Tangerine

The Tangerine[3] component within Citrus enables the labelling of network telemetry incorporating emerging attack patterns, and as a result it produces a labelled intrusion detection data set. This section details the modules which have been implemented within Tangerine to perform such functionality.

```
def run(date):
    ips, data = self.loadTelemetry(date)
    intel = self.intel.lookupIOCs(ips)
    self.intel.parseIntel(intel)
    graph = self.graph.draw(date)
    graphFeatures = self.graph.calculateGraphFeatures(graph)
    kmeans = self.cluster.kmeans(graphFeatures)

    labelledData = self._southbound._cluster.applyLabels(
        data,
```

[3]https://github.com/ruzzzzz/Citrus/tree/main/tangerine

```
        kmeans.labels,
        date
    )
    self._southbound._storage.saveLabelledDF(labelledData, date)
```

Listing 4.1: Driver logic

4.1.3.1 Driver

The *Driver* is the initial entry point to Tangerine, and orchestrates all stages within the labelling process. Listing 4.1 documents the code used in this implementation. The following describes each of these steps in detail. Tangerine starts the process by retrieving the start date from the configuration model within the shared library. This start date is used to begin labelling telemetry collected after a specified date. Tangerine performs the labelling of network telemetry in batches, typically into distinct days of collection. Upon the successful labelling and exportation of a labelled data set, Tangerine increments the start date, and automatically begins to label the next day's telemetry. This process terminates when there is no more telemetry to label, and resumes upon new telemetry becoming available.

When telemetry is available and not already labelled, Tangerine collects it through requests to the Southbound Interface. In particular, the Historic Flow Collector module within the Southbound Interface is utilised. The *Driver* invokes the *query* function within the Historic Flow Collector module, passing the date and data source requested. This module then returns an RDD, which partitions the selected data across nodes within a cluster. This telemetry is then pre-processed through invoking functions implemented within the Data Cleaner located in the shared library. Each distinct IP address is extracted from the telemetry. This is implemented through aggregation operations on the RDD storing the telemetry.

The IP addresses extracted from the previous operations are used to query CTI services. They are passed to the Intelligence Orchestrator, which performs communication with these services. Upon the successful response from a remote service, each response is parsed and objects are constructed in memory, which represent the data contained within. These are implemented as dictionaries, and are used in future operations to produce a graphical representation. The *Driver* then coordinates with the Ground Truth module to draw a graph which represents relationships between potential attackers and entities derived from CTI sources on a specified date. The graph-based features of each node are then calculated. These features are then clustered to identify nodes of similar ilk. Supernodes are discovered from the clusters which represent highly connected nodes. This indicates that they share a relationship with many malicious entities, and can be labelled as such. Nodes which are not part of the supernode cluster are labelled as outliers, as there is no consensus from the intelligence community that they have performed malicious actions elsewhere on the Internet during the period of telemetry capture.

These labels are then applied to the telemetry using the Cluster Operations module within the Southbound Interface. This returns a RDD incorporating emerging attacks, which is supported by a strong ground truth, and known benign communications to internal network services. Additional features are composed before the telemetry is saved, using the Southbound Interface, to HDFS as a labelled intrusion detection data set. The additional features created grant greater insight into the captured data. These include the duration of the flow, as well as the time that the flow started and ended.

4.1.3.2 Intelligence Orchestrator

The *Intelligence Orchestrator* coordinates with Intelligence Service Applications to communicate with various CTI services. The communication with these services can be enabled or disabled using the aforementioned configuration module. Upon initialisation by the Driver module, instances of each enabled application are created.

This module provides *query* and *parse* methods. The *query* method accepts a list of IP addresses, which were observed communicating with a deployed honeypot. Each application instance is then iterated to call the corresponding application's query method, passing the list of IP addresses. After this is done, a list of request objects are returned from the applications which contain the information necessary to query various CTI services. These objects are then transferred through the Southbound Interface to the Intelligence Collector module, which performs the requests and returns each response as a dictionary.

The *parse* method receives these responses. This method must initially deduce which CTI service each response is associated with. The corresponding application's parse method is then called, passing every response that it governs as an argument. Each application then performs specific functionality, parsing the responses into objects within memory, which represent relationships an IP address has with sources derived from threat intelligence services.

4.1.3.3 Intelligence Service Applications

As CTI services provide a non-standard interface, custom applications can be created, which contain unique functionality used to query and parse responses from distinct sources. All applications inherit various methods and variables from a base application class. New applications can be added in a simplistic fashion, as they only require the implementation of *query* and *parse* methods. An example implementation of an application which interfaces with a threat intelligence service providing blocklist information through a remote RESTful API is discussed below.

Each application begins upon execution of its *query*. In the case that authentication is required, the configuration module is invoked to access such information. These are typically usernames, emails, passwords, or a key. The authentication data is then appended to the headers within each request. Through research and experimentation with various CTI service, they typically provide either bulk or individual queries. Bulk queries enable a single HTTP request to look up a number of different IP addresses, while individual queries only permit a single data instance per request. In the case that the CTI service the application is interfacing with allows bulk queries, the payload of the request is modified to contain all IP addresses. Naturally, this reduces the time taken to perform intelligence collection, and is the approach taken if possible. However, for individual queries, a number of request objects must be created with each payload containing a distinct IP address. Upon successful construction, they are returned to the Intelligence Orchestrator.

When responses are received from threat intelligence services, the *parse* method of each application is invoked. Typically, CTI services return responses in JSON format. In order to parse and store this information in memory, the json Python library is used. The status code of the response is initially parsed, and if it is not as expected, the response is dropped. Otherwise, the appropriate data is then extracted from the response. For example every blocklist, including the date, the

IP address is associated with. These are then appended to a dictionary, which is returned to the Intelligence Orchestrator.

The Intelligence Service Applications are implemented in a manner which promotes extensibility, and currently support a diverse range of CTI services. These are further detailed in Table 4.1.

Name	Entity Type	Description
AbuseIPDB	Abuse Report	Allows users to submit reports of malicious actions orchestrated by devices on the Internet.
Apility	Blocklist	Provides the capability to query IP addresses against a diverse range of blocklists.
BGP Ranking	AS	Details historical AS information
Censys	Service	Search engine for Internet connected devices.
GreyNoise	Service	Search engine for Internet connected devices.
Hybrid Analysis	Malware, C&C Servers	Scans uploaded samples and provides access to several extracted IoCs.
Maltiverse	Blocklist	Search engine for Internet connected devices.
OTX	Blocklist	Provides access to custom blocklist by Alienvault.
Shodan	Service	Search engine for Internet connected devices.
ZoomEye	Service	Search engine for Internet connected devices.

Table 4.1: The CTI services used to correlate data points and provide a ground truth

Every CTI service provides access to a varying type of information. For example, the Hybrid Analysis platform [151] provides insight into files identified as malicious, which are associated with an IP address. This service allows users to upload suspected malicious samples. These samples are then scanned by a diverse range of antivirus services to determine whether they perform malicious actions. Hybrid Analysis then extracts IP addresses from the malware through static and dynamic analysis. Hybrid Analysis additionally allows users to query samples based upon the extracted IP addresses through a RESTful API. This is the method in which Tangerine gathers evidence of a node's association with samples classified as malware by third parties. Any other IP addresses associated with the malware sample are also leveraged by Tangerine. These grant additional insight into the infrastructure associated with these samples, such as C&C servers.

```
class AbuseIPDB(Feed):

    def __init__(self, key):

        super().__init__()
        self.url = 'https://api.abuseipdb.com/api/v2/check'
        self.key = key

    def __str__(self):
        return "abuseipdb"

    def parse(self, response):

        if 'reports' in response and len(response['reports']) > 0:
            reports = [report for report in response['reports']]
            ret = {"type": self.__str__(), "data": reports}
            return ret
        else:
            return {"type": self.__str__(), "data": None}

    def query(self, ips):
        queries = []
        for ip in ips:
            rs.append(grequests.get(self.url, timeout=30,  params = {
                'ipAddress': ip,
            }))
        return queries
```

Listing 4.2: AbuseIPDB Application Example

Listing 4.2 documents an example application implemented to coordinate with AbuseIPDB. The *query* method is passed a list of IP addresss as an argument. These are then iterated to create a list of request objects. These objects are identical except for the HTTP parameters, which contain the IP address to query. Once these requests have been sent, the responses are then parsed using the *parse* method. Each of the responses are iterated, and the relevant information is extracted and returned to the driver.

4.1.3.4 Ground Truth

This module extracts the relationships of potential attackers and performs clustering of graph-based features to identify highly malicious nodes. The relationships of these nodes are determined through correlation with the third party CTI services discussed in the previous section. Importantly, services providing access to blocklists and abuse reports are leveraged for their ability to provide rich details about infiltration attempts associated with suspect IPs. To achieve this functionality, this module integrates with various libraries providing graph and clustering capabilities. In detail, these include the NetworkX [152] and scikit-learn [153] Python libraries. NetworkX provides an implementation for the creation and study of graph data structures, including directed graphs and multigraphs, as well as graph-based algorithms such as PageRank. The scikit-learn library is used for the provided k-means clustering algorithm.

The graphing capabilities of NetworkX are utilised to plot every node relationship in a visual network representation. In order to extract the relationships derived through CTI correlation, this module traverses through the dictionaries storing contextual information, which are created by the Intelligence Orchestrator. For every identified relationship, this module updates the graph accordingly. Upon the successful mapping of these relations, built-in NetworkX functions are used to extract graph-based features, which are used in a clustering approach to identify highly malicious nodes. Additional details about the methodology and reasoning behind the choices made to develop the ground truth and the associated labelling approach are outlined in Section 5.2.1.

Each of the modules within Tangerine perform a specific function. When these functions are combined they provide the capability to extract contextual information from third parties and the ability to apply this information on data for labelling purposes. As previously discussed, this is achieved through integration with CTI services and the implementation of graph-based feature clustering. These pieces of functionality are implemented to satisfy Design Requirement 3.

4.1.4 Clementine

Clementine[4] serves as a practical implementation of a framework, which leverages labelled network telemetry to perform emerging attack detection. Clementine integrates with Apache Spark to perform this in an online and distributed fashion. Spark contains streaming functionality which enables the consumption and online processing of network telemetry emanating from devices within the local network. Spark also provides a Machine Learning Library (MLLib), which performs efficient computation of machine learning algorithms on distributed nodes within a cluster. Clementine contains modules which leverage these technologies. The implementation detail of each is outlined below.

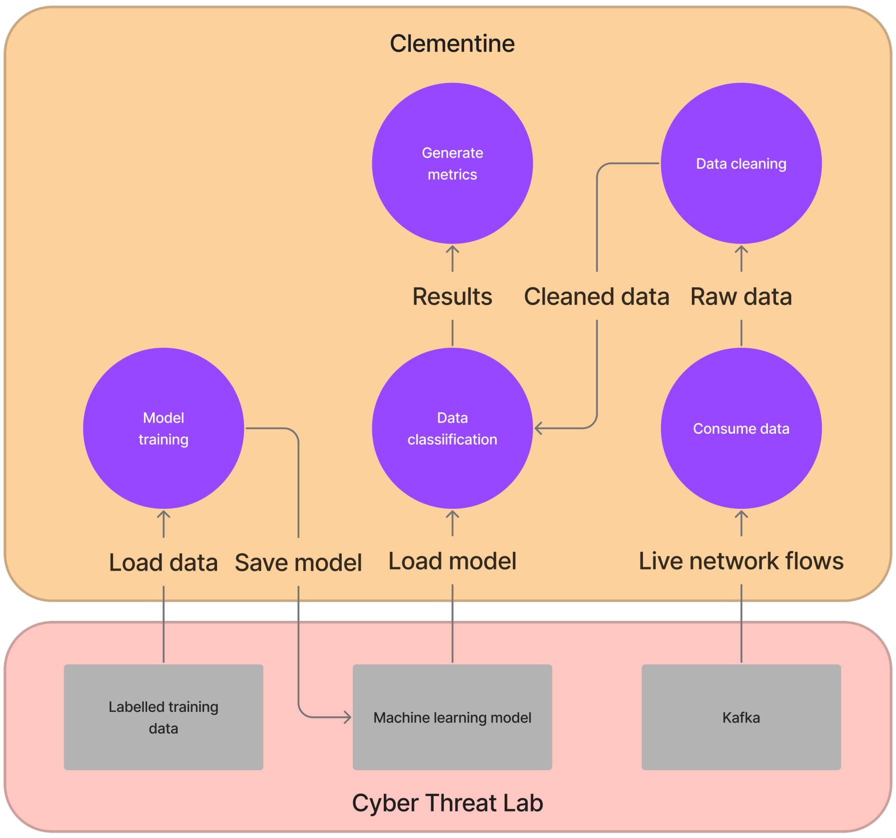


Figure 4.1: Data flow of clementine

[4]https://github.com/ruzzzzz/Citrus/tree/main/clementine

4.1.4.1 Driver

To perform online intrusion detection, the *Driver* must initially load a previously saved model or train a new one, which will be used to predict the intent of each network flow. In the case that a suitable model has not already been created, the labelled data set output by the Tangerine module is partitioned across nodes within a cluster using an RDD. This RDD is passed to the Model Training and Prediction component to train a machine learning model on the data contained within. As illustrated in Figure 4.1, this model is then saved for future use, eliminating the costly training time. In the case that an appropriate model has been previously saved, it is loaded into memory. For each of these scenarios, the Storage Operations component within the Southbound Interface is utilised to load and save objects.

The *Driver* then creates a streaming context, an associated DStream object, along with accumulators which store the counts of the True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) labels of each experimental run. These counts will be used to assess the performance of the detection scheme using widely employed metrics such as accuracy, precision, recall, and F-score. Accuracy is arguably the most intuitive metric, and is simply the proportion of correctly predicted observations over total number of observations. Precision further examines the correctness of positive predictions, and corresponds to the ratio of correctly predicted positive observations to the total number of observations which are predicted to be positive. Recall relates to the proportion of observations predicted as positive over the total number of observations which are positive in actuality. F-score provides a harmonic mean between precision and recall, essentially measuring the balance between the two.

Accumulators are another abstraction provided by Spark, which provide access to shared variables within tasks operated upon by remote nodes within a cluster. In particular, accumulators are variables which can be added through associated operations and are typically used for implementing counters, sums, and other aggregations.

The operations performed on each batch of streaming data are defined using the DStream object obtained previously. This involves pre-processing the network telemetry using the Data Cleaner, predicting the associated label, and calculating performance accuracy metrics. The aforementioned accumulators are used to store the overall count of TP, FP, TN, and FN values. The streaming context is then started, enabling the online consumption of network telemetry.

4.1.4.2 Model Training and Prediction

The *Model Training and Prediction* module contains the functionality to perform ML tasks across nodes within a cluster. This is implemented through leveraging high level abstractions provided by Spark. A critical abstraction leveraged by this module is the ML Pipeline. The ML pipeline specifies a ML workflow through the chaining of *Transformer* and *Estimator* algorithms. A *Transformer* is an abstraction which converts one partitioned collection of elements to another, typically used to append columns such as feature vectors. An *Estimator* abstracts learning algorithms which train on data. This abstraction ultimately produces a model. The ML Pipeline enables the appropriate sequencing of stages required to perform online intrusion detection.

This module implements *train*, *predict*, and *metrics* methods which are leveraged by the Driver. The *train* method is used when a model must be trained on the labelled network telemetry output by Tangerine. A ML Pipeline is created which contains algorithms used to assemble feature vectors and perform supervised machine learning. In detail, these include the VectorIndexer and StringIndexer transformers, which construct feature and label vectors respectively. Additionally, an estimator which performs classification is added to the pipeline. The specific estimator varies depending upon which classification algorithm is intended to be evaluated. The Pipeline's *fit* method is called in order to execute each stage in the pipeline and ultimately train the model, which is then returned to the Driver.

The *predict* method implemented within this module uses the model created previously to predict the label of network telemetry. The telemetry from each streaming batch is fed through the same ML pipeline to ensure that training and prediction features encounter identical steps. The labels are appended to the input RDD, and returned to the Driver.

The *metrics* method leverages MLLib's Evaluation abstractions to gain performance accuracy metrics. These are generated for each batch of streaming telemetry. As mentioned previously, accumulators are used to count the total TP, FP, TN, and FN values for each experiment.

All of the functionality discussed above enables the distributed processing of large amounts of data across a number of compute instances. This functionality is implemented in such a manner to perform intrusion detection of network traces using a range of ML algorithms. This has been implemented to satisfy Design Requirement 4.

4.2 Intelligence Sharing

To enable the research community to replicate the intrusion detection capabilities and achieve similar results, the data set output by the Tangerine module is periodically released to a public GitHub repository [154]. It is also envisioned that next generation intrusion detection mechanisms can utilise this data set for evaluation purposes. A Python script is developed which automatically performs this functionality.

The script initially checks to see if any new telemetry has been labelled. This is achieved through a library which enables access to HDFS operations within Python, hdfscli [155]. If new telemetry is discovered, the library is used to retrieve it locally. The directory structure of the repository is then amended to reflect the changes. Each new file is then compressed using the standard zipfile Python library. This heavily reduces file size, as the data set is released in a CSV format. Once all new telemetry has been retrieved and compressed, changes are commited to the remote GitHub repository, and subsequently pushed to periodically release the data set.

4.3 Summary

This section has highlighted the implementation detail of Citrus and additional tools. The functionality of Citrus is encapsulated within two modules: Tangerine and Clementine. Tangerine performs automated labelling of network telemetry to produce a labelled intrusion detection data set. Clementine utilises this data set to perform online intrusion detection. In order to deal with the scale at which traffic is required to be processed in modern networks, Clementine integrates with Apache Spark, a cluster computation and parallel processing framework. To share intelligence gathered with the research community, an additional script is developed, which automatically releases each output data set to a remote GitHub repository. In the following section, several aspects of the implementation are evaluated to determine whether Citrus has achieved its design requirements.

Chapter 5

Evaluation

In this chapter, the implementation of Citrus is evaluated. Multiple experiments are performed which aim to evaluate Citrus in varying manners. Each evaluation is also performed with the intention of discovering whether Citrus has met the aforementioned design requirements, as outlined in Section 3.1, which are motivated by challenges identified within research.

Initially, in Section 5.1, the evaluation environment is discussed, highlighting an appropriate network architecture suitable for Citrus' operation. Subsequently, the telemetry captured from services deployed within this architecture is examined in Section 5.2, performing analysis and detailing the properties of the corresponding intrusion detection data set authored by Citrus. As outlined in Section 2.1, the modern threat landscape consists of diverse evolving attack patterns. This evaluation will critically examine the attack patterns incorporated within the data set. In addition, a comparison between several other intrusion detection data sets within literature is done to showcase its beneficial properties. Critically, this evaluation will assess whether Design Requirements 1 and 2 have been achieved by Citrus, and will explore aspects of realism and whether the attack data is representative of the modern threat landscape to accomplish this. The lack of these properties in relation to intrusion detection data sets has made the comprehensive evaluation of detection mechanisms extremely problematic within literature.

Section 5.3 evaluates the ground truth developed by Citrus. In order to understand whether Design Requirement 3 has been met, the developed ground truth must be validated to ensure its accuracy. This evaluation divulges intricate details about the methodology behind the development of the ground truth, and further provides results which are intended to showcase the appropriateness of Citrus' method using established cluster-based metrics.

Research Challenge 3 states that traditional approaches to anomaly detection are not appropriate for the vast amount of data within modern networks. In order to provide contributions within this area and help alleviate this challenge, Citrus must achieve Design Requirement 4. This will ensure Citrus is capable of providing accurate detection of emerging attacks in a practical real-world environment. In addition, Citrus must also provide a detection approach which is able to handle such large volumes of data, and predict in near real-time to provide network administrators timely alerts relating to intrusion attempts. It will be assessed whether Citrus has met Design Requirement 4 through two separate evaluations, which are further discussed below.

Section 5.4 evaluates the detection capabilities granted by Citrus. Initially, this is achieved through the exploration of various machine learning algorithms and their corresponding accuracy to detect attacks within the data set produced by Citrus. The models created by these algorithms are trained offline, and are also originally used in an offline manner to predict unseen attack data. The results extracted from this evaluation are used to inform the most suitable algorithm for online intrusion detection scenarios. Also included in this evaluation are experiments which assess Citrus' ability to detect attacks in a practical manner using a real network environment. Several scenarios are outlined which use machine learning models, which are trained offline, in an online fashion to predict live data emanating from devices within the network.

Finally, Citrus' ability to efficiently process vast amounts of data is discussed in Section 5.5, specifically exploring the benefits provided through integration with frameworks providing parallel processing capabilities. This evaluation uses the data pipeline leveraged by Citrus to detect live attacks. This will assess whether Citrus' intrusion detection capabilities are suitable for the substantial amount of data within modern networks and whether near real-time prediction is possible.

5.1 Evaluation Environment

In order to evaluate Citrus, a suitable network environment is required. A research facility located within Lancaster University, the Cyber Threat Lab [156], is used for this purpose. The laboratory is designed to provide a collaborative platform that enables the analysis of emerging threats in a controlled environment. Consisting of multiple inter-connected components, the Cyber Threat Lab grants access to a plethora of malicious data garnered by a myriad of sources. This data proves fruitful in understanding the manner in which attackers operate and is ultimately used in this thesis to detect such attacks. This facility is physically composed of five servers, each containing an Intel Xeon E5-2620 v3 processor and 128GB RAM. The VMWare [157] hypervisor is used to manage virtual machines within this environment.

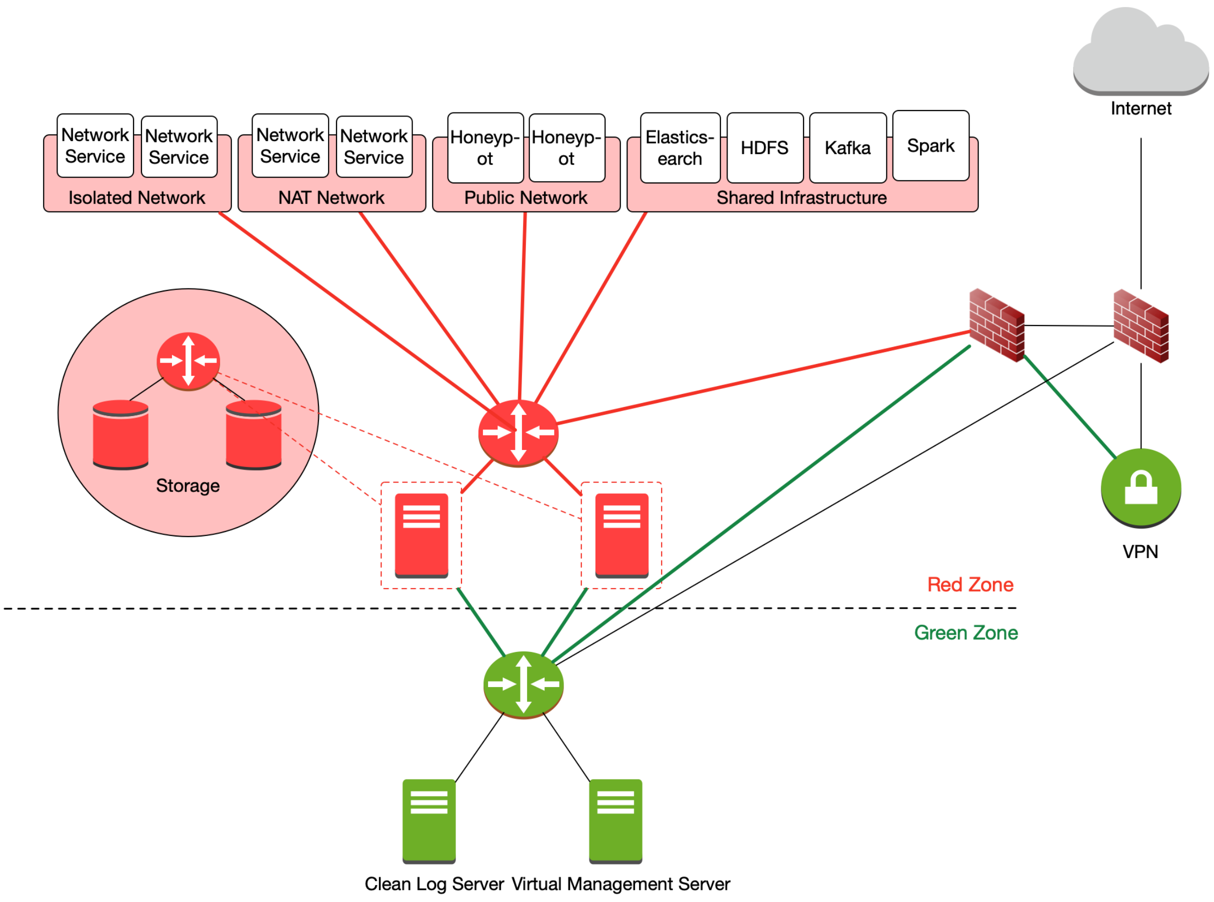


Figure 5.1: Cyber Threat Lab Architecture

As illustrated in Figure 5.1, there are two primary logic zones within the Cyber

Threat Lab, a green zone and a red zone. The green zone allows management and entry into the environment, and the red zone hosts the machines and devices in which network services are deployed.

Within the red zone, there exists a number of segmented networks, which employ varying access to local and internet services. *Public networks* are designed to emulate servers and devices that are directly publicly accessible over the Internet. This enables the analysis of emerging attacks that are typically conducted against vulnerable network services at scale. For the evaluation of Citrus, various honeypots are deployed within this network in order to attract adversaries to perform malicious actions.

Citrus also utilises services deployed within the *Shared Infrastructure*, which allows other red zone networks to utilise central infrastructure such as logging, authentication, name resolution, etc. Additional network services are required for the operation of Citrus, and the deployment is discussed in more detail in Section 5.1.1.

NAT Networks are designed to emulate personal, enterprise or organisation networks. Internal hosts are all permitted outbound access through a shared public address.

Isolated networks are used to monitor activity that do not require any external connectivity outside of the lab or have a significant risk of negatively impacting external sources outside of Lancaster University. This is especially useful for the analysis of malware garnered from monitoring the experiments in the other networks.

5.1.1 Deployed Network Services

Citrus requires various services to be deployed within the network. A range of these services have been instantiated within the Cyber Threat Lab, which support Citrus and the work in this thesis in general. Notably, Citrus requires telemetry which incorporates malicious and benign activity. As motivated by related work in this area, honeypots are utilised to capture a plethora of emerging attacks. These are deployed within the Public Network segment of the Cyber Threat Lab. This grants each deployed honeypot a WAN IP address, which is accessible from the Internet.

A variety of different honeypots are deployed, which emulate various vulnerable services, including TPot [158]. TPot is composed of a number of medium interaction container-based versions of popular honeypots, which span multiple protocols. Consisting of over fifteen honeypots in total, TPot enables the capture of a wide variety of emerging attack telemetry, including attacks targeting ICS devices. Additional standard versions of honeypots are scattered throughout the Public Network, including the aforementioned Dionaea and Cowrie honeypots. Critically, every deployed honeypot is either medium or low interaction to ensure the integrity of the Cyber Threat Lab. These types of honeypots contain exposed ports, which emulate popular services that are also vulnerable to a range of vulnerabilities. As they are emulated, in reality the servers are not susceptible to such vulnerabilities. As such, these honeypots are to able to securely store the traffic characteristics of attack attempts directed towards Lancaster University's public address space.

The telemetry data associated with every deployed honeypot is captured by Cisco's Joy [159] tool. Joy provides a libpcap based software solution for the extraction of data features from live network traffic. This is achieved using a flow oriented model, and is represented as JSON. This telemetry is transferred to a distributed database located within the Shared Infrastructure network segment. The database deployed within the Cyber Threat Lab is Elasticsearch, a distributed NoSQL database suitable for the storage of JSON based network telemetry. The telemetry is indexed within the database based upon the date of collection. Elasticsearch is deployed within the network in a distributed fashion. A cluster is formed, which is comprised of two master and three data nodes. Elasticsearch also provides a convenient agent, Logstash [160], which is used to transfer all telemetry from the honeypots to a database instance.

As one of the requirements of Citrus is to perform intrusion detection by leveraging machine learning algorithms, it is also necessary to capture benign telemetry. This provides a profile of normal behaviour, which is used by such algorithms to correctly identify flows which do not pose a threat. This known benign traffic is captured from internally accessible VMs located within the Shared Infrastructure. In a similar vein to the approach taken by Song et al. [18], these VMs perform two main functions, acting as a DNS server and a data node. These servers are also instrumented with additional communication protocols, e.g. ssh, for management purposes. The telemetry is extracted from these services in the same manner as the honeypots, leveraging Joy and the Logstash agent. Furthermore, all traffic related to these network services are regarded as benign as there were no observed attacks within this controlled environment.

A Hadoop cluster is deployed within the Shared Infrastructure to store the labelled intrusion detection data set output by Tangerine in a format suitable for sharing with the research community. This service is also used by Clementine to gain rapid access to the data set to perform intrusion detection. This cluster consists of one master node and six data nodes.

In addition, Citrus integrates with Spark to perform tasks in parallel on nodes within a cluster. A Spark cluster is deployed within the Shared Infrastructure to orchestrate and execute these tasks. Each of the six nodes within this cluster are provisioned with 4 vCPU cores and 64GB RAM. Citrus utilises Hadoop YARN as a cluster manager to schedule tasks upon the executor nodes.

Finally, a cluster of Kafka brokers are deployed within the network to transfer live network telemetry to Clementine. This is consumed by the Spark Streaming framework to perform intrusion detection on batches of streaming flows. To provide fault tolerance, three brokers are deployed within the cluster. These are provisioned with 4 vCPU cores and 12GB RAM.

Each of the services residing within the Shared Infrastructure are accessible through a specific private IP address and port. In order to enable the integration of Citrus with other network environments, these IP addresses and port numbers are specified within a configuration file.

5.2 Data Set

The telemetry captured within the Cyber Threat Lab is compiled to create a flow-based intrusion detection data set with a robust ground truth. In this section, the properties of all telemetry captured within the operational period is presented. Additionally, analysis is performed to reveal statistical properties of the data.

The operational period, in which automatic network telemetry collection and labelling is conducted, initiated in June 2020. Due to the automatic nature of this process, there is no fixed end date. As a result, the intrusion detection data set compiled from this telemetry will receive periodic updates for the foreseeable future. To enable the analysis of the data set, records are used up until 15th December 2020 in this evaluation. This novel intrusion detection data set, which is produced by the Tangerine component within Citrus, is named LUFlow '20. LUFlow '20 is released to the general public through a GitHub repository [154]. This release anonymises IP addresses to alleviate privacy concerns.

The remainder of this section showcases data analysis performed on the LUFlow '20 data set. As previously discussed, this data set is constantly updated to reflect new attacks captured by honeypots deployed within the Cyber Threat Lab. Therefore, the results obtained from any future analysis performed may be subject to change. However, this analysis is provided to supply researchers with an insight into the threat landscape at the time of publication.

Given that the placement strategy of the honeypots ensures that they are accessible from the Internet, several types of network traffic are observed. This includes scanning activity, noise, and malicious behaviour. The honeypots are also placed within IP address space belonging to a University. University networks often allow a diverse range of services and applications to be used. This property could be very attractive to an adversary since infiltration could open up other avenues to attack. As a result, it should be noted that the data encompassed within this data set may not be observed in other non-academic networks.

It should also be made clear that despite the data set receiving updates through automatic releases, the data set and environment have not been maintained since 2021. It is now not known what will occur within the environment and how that would affect the data capture. Therefore, there is no guarantee that the traffic meets the assumptions stated in the original capture design.

5.2.1 Overview

	Number of flows	Mean flows per day	Percentage
Total	166,815,387	926,752	100%
Benign	90,852,768	504,737	54.46%
Malicious	57,922,695	321,792	34.72%
Outlier	18,039,924	100,221	10.81%

Table 5.1: Distribution of flow labels within LUFlow '20

Table 5.1 provides an overview of the distribution of labels within LUFlow '20. During the selected period of observation, there was a total of 166,815,387 flows captured within the Cyber Threat Lab, of which, 90,852,768 flows are known to be benign. As previously discussed, all traffic relating to privately accessible internal network services is regarded as benign. The telemetry captured through the composition of honeypots within the Cyber Threat Lab is subject to Tangerine's labelling mechanism. This mechanism identifies malicious nodes through correlation with third party CTI services. The number of malicious flows labelled in this manner is 57,922,695. The nodes identified as having no association with malicious entities through this correlation process, have their corresponding telemetry labelled as an outlier. The total count of these outlier flows within LUFlow '20 is 18,039,924. These flows remain in the data set to encourage the practice of manual analysis to determine the true intent behind the unsolicited form of communication.

5.2.2 Geolocation Analysis

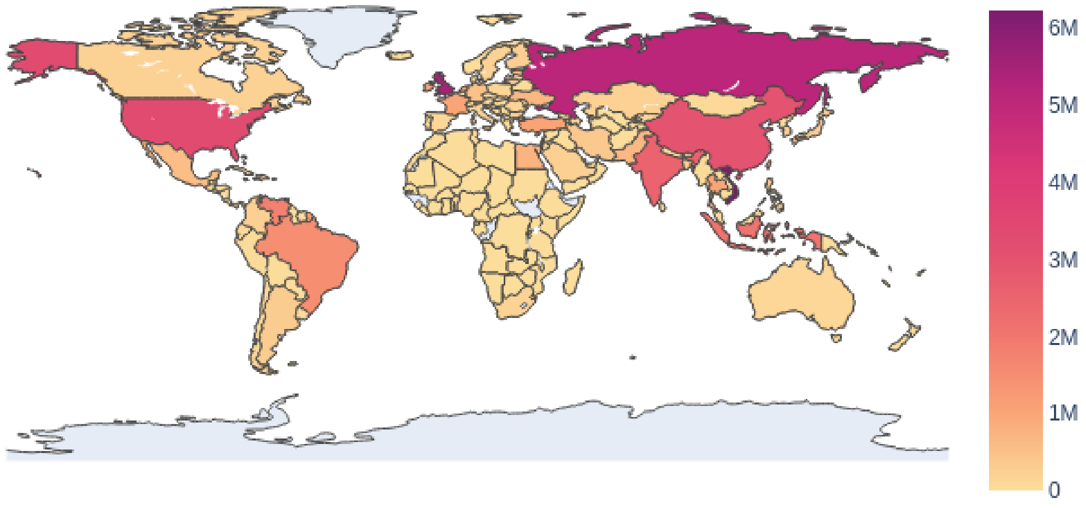


Figure 5.2: Number of flows distinguished by geographic location

The results outlined in Table 5.2 showcase the locations of the top 10 source IP addresses identified in the telemetry captured by nodes within the Cyber Threat Lab. This analysis is included to highlight potentially untrustworthy regions or ASs. In order to map an IP address to an ASs, the GeoLite2 ASN database [161] is used. The IP addresses of these servers have been anonymised to consider the privacy of the individuals. As shown, a server within AS49509, an ISP in Russia, initiated the most flows by a substantial margin. Interestingly,

IP Address	Flow Count	ASN	Country
x.x.x.1	810,793	49505	RU
x.x.x.2	424,132	49877	RU
x.x.x.3	423,777	37963	CN
x.x.x.4	397,955	207566	RU
x.x.x.5	279,090	213371	NL
x.x.x.6	243,113	213371	NL
x.x.x.7	234,692	43350	NL
x.x.x.8	233,956	49877	RU
x.x.x.9	225,281	49877	RU
x.x.x.10	221,126	43350	RU

CN = China, RU = Russia, NL = Netherlands

Table 5.2: The top 10 source IP address locations

Russian ASs are the source of over half of the top 10 IP addresses seen in this analysis. Furthermore, AS49877 appears in three separate instances within this table. This AS is associated with a hosting provider serving the Russian and Moldovan regions.

The geographic distribution of all flows associated with the deployed honeypots, i.e. malicious or outlier flows, is illustrated in Figure 5.2. This is intended to visualise the geographic origin relating to potential infiltration attempts. In total, there are flows associated with 209 distinct countries within LUFlow '20. In this investigation, it is observed that flows relating to servers originating in Russia are the most prevalent. It is also observed that around 50% of all unsolicited flows captured by honeypot telemetry originate from five countries: China, Vietnam, United Kingdom, United States, and Russia.

Furthermore, there are only a handful of countries which do not appear within LUFlow '20. These include small countries within the continent of Africa, as well as Greenland and Antarctica. Notably, despite harsh restrictions and censorship on the Internet, servers designated as originating from North Korean ASs are identified as interacting with the honeypots on multiple dates.

5.2.3 Source IP Address Analysis

Figure 5.3 illustrates statistics regarding the number of source IP addresses identified within the telemetry. The purple line represents the occurrence of unique

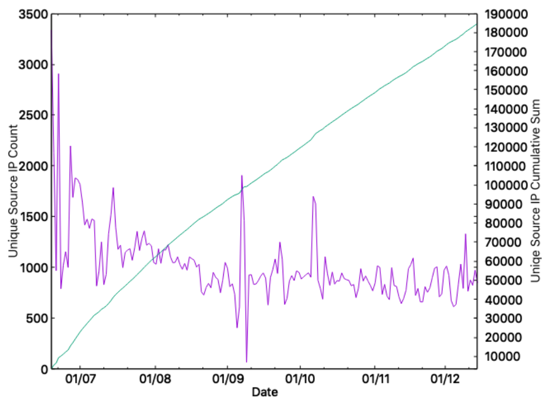


Figure 5.3: Distinct count of source IP addresses per date

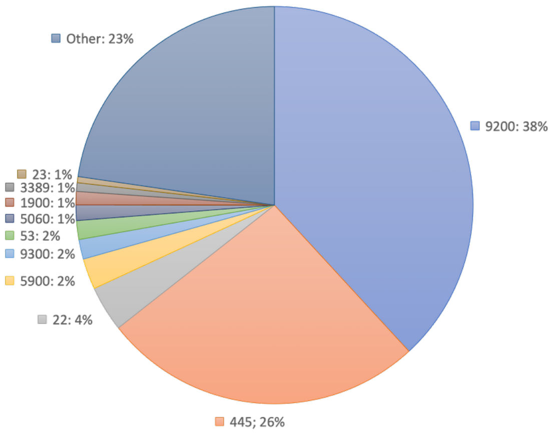


Figure 5.4: Distribution of destination ports

source IP addresses in each day, while the green line represents the accumulation, i.e. the cumulative sum, of these unique IP addresses identified. As evident within this figure, the total number of unique Source IP Addresses within LUFlow '20 is 184,751. This overall count is steadily increasing, with an average number of 1,026 new source IP addresses being discovered within the telemetry every day.

However, there is an evident negative trend, with the number of new source IP addresses gradually reducing for every day of telemetry capture. This is suggested to be the result of adversaries exhaustively enumerating every possible attack vector. Upon the unsuccessful infiltration of the deployed honeypots, attackers will typically move on to the next target.

5.2.4 Destination Port Analysis

Figure 5.4 depicts the various destination ports which have been used to communicate with services within the Cyber Threat Lab. These include services which have been targeted by attackers, as well as network services used to profile normal behaviour. The LUFlow '20 data set contains flows which explore every available port, ranging from 0 to 65535. As evidenced in the figure, the destination port 9200 occurs the most commonly within LUFlow '20. This port is frequently used to communicate with the distributed database, which stores the telemetry used to compile LUFlow '20, and as a result is mainly used for benign purposes. However, not all traffic destined to this port represents normal behaviour. The Elasticpot

[162] honeypot is deployed within the Cyber Threat Lab using the aforementioned TPot platform. This honeypot emulates various vulnerabilities inherent wihin Elasticsearch instances, and as a result attracts malicious behaviour on the same port. However, when filtering out normal flows within the data set, the number of flows to this destination port become substantially less.

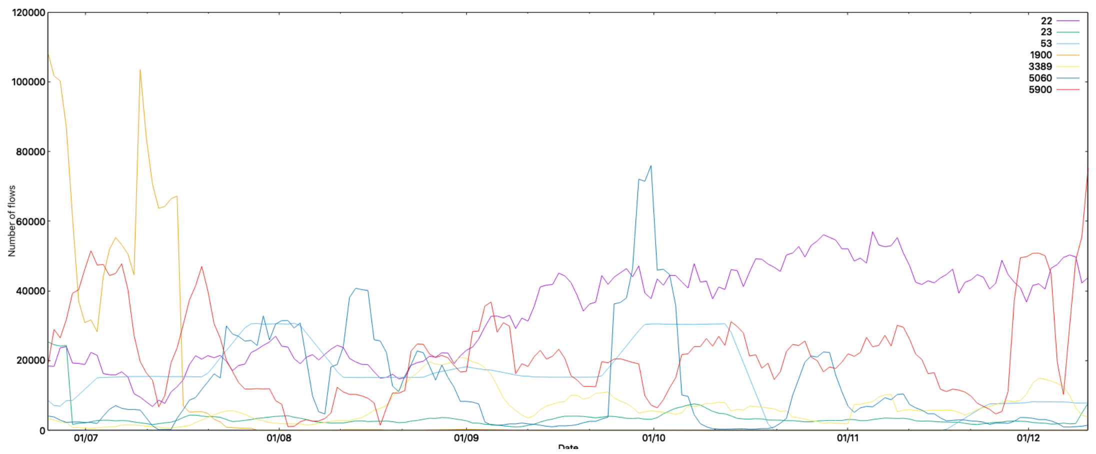


Figure 5.5: Number of flows to distinct services exposed by honeypots

The next most prevalent port within the data set is 445, which is typically used by SMB services. This service has received patches, which aim to fix critical vulnerabilities, such as RCE . This investigation has identified a number of these vulnerabilities, which are still being actively exploited in the wild. Most notably, the infamous Eternalblue exploit (CVE-2017-0144), which has been previously discussed in the related work section, has caused devastating damage through the incorporation into malware and botnets such as WannaCry and Petya. Due to the powerful nature of these exploits, they are still propagating at an alarming rate, as evidenced by the high number of requests to this port within LUFlow '20.

In order to further examine how attack patterns evolve over time, a time-series of the number of flows directed towards services exposed to the Internet by honeypots deployed within Lancaster University's network address space is included in Figure 5.5. The most popular services, as identified in Figure 5.4, are included in this figure, with exception of port 445 due to it receiving substantially more flows than the others, which makes the measurements of the remaining services less legible. The values included within the figure represent a seven day moving average, a common statistical technique used to smooth any short-term fluctuations and highlight long-term trends inherent within the data. This is done due to the existence of highly volatile measurements observed for the services on each date.

This time-series enables the identification of a number of trends, which correspond to potentially new attack vectors or threat actors. Notably, the destination port 5900 is also heavily targeted towards the end of 2020. Further investigation has revealed that this port corresponds to implementations of the Virtual Network Computing (VNC) service, which enables the remote control of another computer. Though the true nature behind the reason this service is highly targeted by attackers is unknown, there has been a proliferation of novel vulnerabilities aimed at this service. These include powerful exploits, such as CVE-2020-14404, which enable malicious actors to remotely execute code in order to infect devices en masse and perform further malicious behaviour [163]. Nevertheless, it is suggested that the large amount of telemetry directed at this port may be indicative of large scale malware campaigns in which this service is targeted to gain a foothold into infrastructure.

In addition, there are an increased number of requests towards port 53, which is used to handle name server resolutions, towards the end of July 2020. One possible explanation for this is the public disclosure of CVE-2020-1350 on the 14th of July, a vulnerability discovered within Windows Domain Name System servers [164]. This vulnerability enables unauthenticated adversaries to execute arbitrary code in the context of the Local System Account. The vulnerability has also been given the maximum severity rating of 10, indicating that it can be heavily exploited to infiltrate a large number of remote servers. As usual for a vulnerability of this nature, several Proof of Concept (PoC) exploits were released on code sharing platforms the next day [165]. Naturally, malware authors could easily incorporate this attack vector into their arsenal within the following days, which could be the reason behind the increased measurements relating to port 53 observed in Figure 5.5.

A large number of requests is also observed towards port 1900 at the start of July 2020. Any traffic to this port is suspicious, as it has no association with the benign behaviour profile inherent within the data set. Interestingly, traffic related to this port drops substantially in the middle of July. This trend has continued towards the end of the period of observation, with an average number of flows related to this port being around only 20 every day for the month of December. Port 1900 is used by services offering Universal Plug and Play (UPnP) functionality, and has in the past been plagued by code execution exploits such as CVE-2018-16596. It is possible that all vulnerable servers have been exhaustively discovered, attacked, or patched. In this case, it may not be lucrative enough for malware authors to use these exploits anymore. Another possible explanation could be the collapse of botnets and their associated infrastructure, which have previously leveraged this vulnerability.

Another observation is the general upwards trend of traffic relating to port 22. This port is exclusively reserved for the SSH protocol. SSH is an industry standard protocol used to remotely instrument remote servers through the command line. The consistently high number of flows relating to the SSH service could be indicative of the number of unsecured, or loosely secured, SSH servers that are typically provisioned with newly instantiated servers on the Internet. Attacks which target this port usually leverage a brute force method to search for possible common username and password combinations.

As evidenced within this analysis, the services targeted and the corresponding type of attacks orchestrated against the deployed honeypots vary over time. Due to the evolving nature of attack procedures, it is essential to keep abreast of emerging attack patterns which are likely to be used against critical infrastructure. Defense mechanisms such as IDSs require attack data which is representative of these potential attacks in the form of signatures or training data. As LUFlow '20 is continually updated to reflect these emerging attack patterns, training data used in anomaly detection approaches and signatures used in misuse detection approaches can be consequently updated to defend against similar emerging variants.

5.2.5 Extracted Features

As mentioned previously, each row within the LUFlow '20 data set represents a distinct network flow captured within the Cyber Threat Lab. Each column within the data set represents a feature which describes a certain facet of the flow. Inspired by predecessor intrusion detection data sets [18, 19, 101, 104, 73, 91], LUFlow '20 contains a variety of significant features extracted from both benign and malicious flows. These features, which are recorded in Table 5.3, incorporate both packet-based and flow-based features. Each flow is defined as a set of packets with common characteristics. In this instance, the conventional

#	Name	Description
1	src_ip	The source IP address associated with the flow. This feature is anonymised to the corresponding Autonomous System.
2	src_port	The source port number associated with the flow.
3	dest_ip	The destination IP address associated with the flow. The feature is also anonymised in the same manner as before.
4	dest_port	The destination port number associated with the flow.
5	protocol	The protocol number associated with the flow. For example TCP is 6.
6	bytes_in	The number of bytes transmitted from source to destination.
7	bytes_out	The number of bytes transmitted from destination to source.
8	num_pkts_in	The packet count from source to destination.
9	num_pkts_out	The packet count from destination to source.
10	entropy	The entropy in bits per byte of the data fields within the flow. This number ranges from 0 to 8.
11	total_entropy	The total entropy in bytes over all of the bytes in the data fields of the flow. This number ranges from 0 to 8n, where n is bytes_out plus bytes_in.
12	mean_ipt	The mean of the inter-packet arrival times of the flow.
13	time_start	The start time of the flow in seconds since the epoch.
14	time_end	The end time of the flow in seconds since the epoch.
15	duration	The flow duration time, with microsecond precision.
16	label	The label of the flow, as decided by Tangerine. Either benign, outlier, or malicious.

Table 5.3: The various features inherent within LUFlow '20

network five-tuple is used: source IP address, source port number, destination IP address, destination port number, and protocol. Furthermore, bidirectional flows are created by combining unidirectional flows which are part of the same session. Bidirectional flows consist of a pair of unidirectional flows whose source addresses, destination addresses and ports are reversed. This enables both inbound and outbound communication within a single flow. Critically, LUFlow '20 provides a ground truth through flow labels. Since all attacks are real, i.e. they are not injected into the data set, it is not possible to accurately label each *type* of attack. Therefore, the target labels of each flow are considered to be either benign, outlier, or malicious.

5.2.6 Data Set Comparison

A comparative analysis is conducted between LUFlow '20 and other related IDS data sets surveyed in recent literature. Only publicly available data sets which incorporate a ground truth in the form of target labels are considered in this comparison. Table 5.4 documents the properties of each of these data sets.

Name	No. of networks	No. of distinct IPs	Simulation	Attack Injection	Duration	Updated
KDD '99 [19]	2	11	Yes	Yes	5 Weeks	No
MAWILab [114]	1	Unspecified	No	No	19+ Years	Yes
Kyoto 2006+ [18]	5	4,420,971	No	No	2 Years	No
ISCX 2012 [123]	4	21	Yes	Yes	7 Days	No
CTU-13 [124]	2	Unspecified	No	Yes	6 Days	No
UNSW-NB15 [101]	3	45	Yes	Yes	16 Hours	No
CICIDS2017 [100]	5	500	Yes	Yes	1 Week	No
LUFlow '20 [154]	4	184,751	No	No	6+ Months	Yes

Table 5.4: Comparison of IDS data sets

In summary, the key differences which separates LUFlow '20 from the majority of other IDS data sets is the *real* nature of network traffic and incorporated attacks, which reflect emerging threats currently propagating. As detailed in Section 3.1, the design of Citrus ensures *real* benign and malicious traffic are constantly captured, labelled, and included in periodic releases of LUFlow '20. The majority of data sets surveyed in literature are created using simulation tools, which fundamentally generate synthetic network traces. As identified in related work, data sets who choose this method face challenges such as seamlessly blending benign traffic with malicious traffic, which inhibits the utility and reliability of the data set to accurately represent real-world intrusion scenarios. Additionally, LUFlow '20 does not contain any attacks which have been manually injected into the data set. This ensures that all of the malicious activity incorporated within it is truly representative of emerging attack vectors.

Furthermore, LUFlow '20 is the only *recent* data set which receives constant updates. As documented in Table 5.4, the MAWILab data set is updated in daily intervals. However, the labelling mechanism was implemented based upon the output of the combination of four dated unsupervised learning algorithms. During the period between the detector implementation and present day, there has been a substantial evolution in the way in which attacks manifest themselves, which suggests this approach is no longer relevant. As other researchers have indicated, this approach lacks accuracy for modern day attack vectors and network traffic in general [97]. This becomes evident upon manual inspection of the data set, with many anomalous traffic traces exhibiting normal patterns.

Fundamentally, LUFlow '20 was created with the intention to constantly capture network traces incorporating real attacks and normal behaviour using a robust ground truth. Notably, this is the first data set made publicly available which considers the development of a ground truth through active CTI collection. The constant nature of this telemetry capture and labelling through correlation with CTI services enables the data set to be continuously updated. To the best of our knowledge, an updated data set has not previously been achieved through the use of honeypots. Critically, this allows LUFlow '20 to reflect novel threats encountered in the wild. Moreover, this constant feed of labelled network telemetry helps alleviate current problems facing the research community, such as the over study of data sets, or publication of irrelevant results on outdated data sets [23].

LUFlow '20 also maintains a good balance of malicious and benign traces. This is in stark contrast to the CICIDS2017 data set, which has a high class imbalance. High class imbalance influences classification algorithms to be biased towards the minority class, thus, resulting in lower accuracy. In the case of CICIDS2017, the benign class accounts for 83% of the total data. As identified by Panigrahi et al. [126], random samples taken from this data set often result in a large number of attacks being omitted from training data. LUFlow '20 incorporates millions of malicious traces, and as discussed in Section 5.2.4, it also includes a wide range of different attack techniques. This ensures classification algorithms that use LUFlow '20 as training data have access to rich attack telemetry.

The Kyoto 2006+ data set also provides an insight into the geographic distribution of servers which communicate with their honeypots. They report that 50% of attacks are initiated by servers located in China, United States, and South Korea. As discussed in Section 5.2.2, 50% of malicious LUFlow '20 data relates to traffic from China, Vietnam, United Kingdom, United States, and Russia. This shift in geographic distribution could be influenced by a number of factors, including the fact there is over a 10 year gap between the data sets. As well as this, there exists differences between the environment in which data was captured. Despite both environments belonging to a University, Kyoto 2006+ was captured within Japan, while LUFlow '20 was captured in United Kingdom. This could help explain the large number of malicious flows emanating from devices in United Kingdom for LUFlow '20 and the lack of these in Kyoto 2006+.

5.3 Ground Truth Development

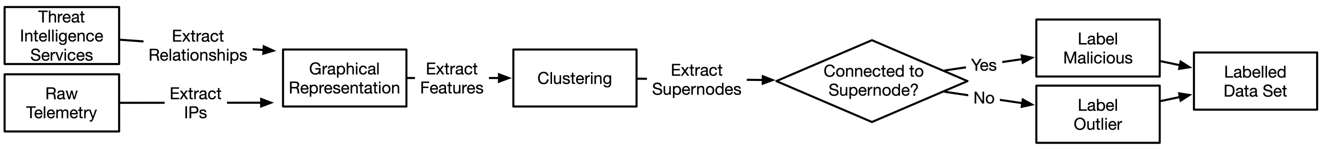


Figure 5.6: From raw telemetry to labelled data set

5.3.1 Methodology

Based upon the restricted ground truth identified in literature, an emphasis has been placed upon the development of a robust ground truth. Citrus' Ground Truth module contains the functionality required to map any identified relationships relating to remote servers which interact with deployed honeypots in the Cyber Threat Lab. These relationships are then used to derive the ground truth. A high-level overview of this labelling process is presented in Figure 5.6.

Initially, the raw flow measurements are collected, and the unique IP addresses are extracted. CTI services are then queried, using the IP addresses as parameters. This step is taken to correlate suspect activity with third parties. The data collected from these services is then processed, and each record's date is compared against the date the captured telemetry occured. Records with matching dates are then extracted and used for plotting within a graph. This process ensures derived relationships are valid for the suspected attacker on the date they interact with deployed honeypots.

This network of relationships derived from third party CTI services is defined as an undirected graph, which is composed of nodes connected by edges. In this graphical representation, nodes can be connected to any number of other nodes, however, they cannot be connected through an edge to themselves, or contain parallel edges to the same node. Formally, the graph is defined as $G = (V, E)$, where V is the set of vertices, and E is the set of edges.

In this instance, the nodes within the graph are the suspected attackers IP addresses, which are connected through edges to entities derived from the collected CTI. These connected entities represent all known associations of a suspect attacker present on the date of the telemetry capture. Therefore, this method identifies nodes which have been observed to be performing malicious actions on the date the telemetry was captured.

Due to the diversity and heterogeneity resulting from the variety of CTI services utilised by Tangerine, each service provides an edge between a suspect attacker, u, and a variable entity. These entities currently represent blocklists, v, malware samples, w, Autonomous Systems, x, (through ASNs), and available services, y. Considering this, the set of vertices in G is defined as:

$$V = \{u_1, ..u_n, v_1, ..v_n, w_1, ..w_n, x_1, ..x_n, y_1, ..y_n\} \tag{5.1}$$

and there exists an edge from u to v, w, x, or y if the CTI collected indicates a connection between them.

The edges indicate a relation between a server which initiated communication with a honeypot and an entity as derived through CTI correlation. For example, a suspected attacker may be using a network interface, which has an IP address that belongs to an AS. In this case an edge would connect the attacker to this AS. This suspected attacker may also have been observed performing malicious behaviour elsewhere on the Internet, and as such they have been placed on a blocklist. Through correlation with CTI services, which provide blocklist data, an edge would also be made between the suspected attacker and a given blocklist. The significance of this is that because the suspected attacker has been placed on a blocklist, the traffic captured by the honeypot is also likely malicious. Furthermore, suspected attackers that share a large number of common associations,

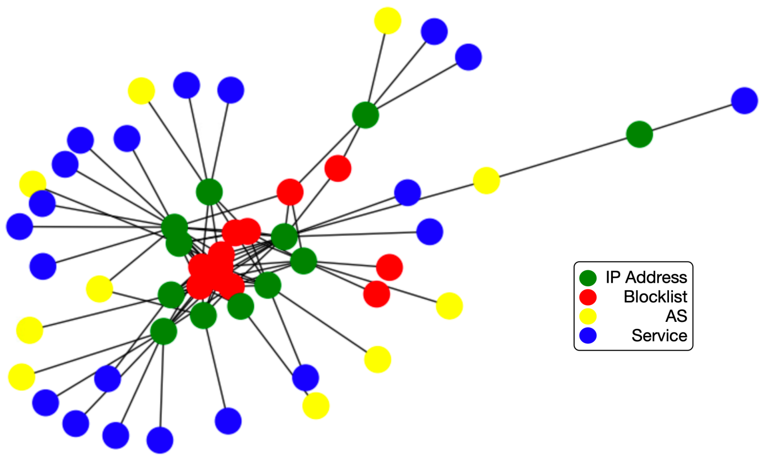


Figure 5.7: Subgraph example of node relationships

such as blocklists or malware samples, can be viewed as highly malicious and further increases the possibility that the data captured by the honeypots is also malicious. The intention of this approach is to identify these highly malicious individuals.

Each edge between a suspect attacker and an entity has an associated weight, which distinguishes its importance. For example, a suspect attacker having an edge with a blocklist is of high importance in the determination of malicious intent, and therefore has a high weight within the graph. On the other hand, a suspect attacker having an edge with an AS is expected, and as such requires a much lower weighting.

For example, Figure 5.7 illustrates a small sub graph used to highlight the approach. Blocklists, denoted by red entities, reside in the center and are connected to IP addresses, green nodes, which are active within the blocklist on the date of telemetry capture. These nodes are then also connected to an ASN, yellow entities, and exposed services, blue entities. Exposed services represent a port open on a server. In order to identify nodes which belong to a large number of malicious entities, such as blocklists, features are extracted from the graph.

The features used in this approach include node degrees and eigenvector centrality. For a particular node in a graph, degrees represents the total number of connected edges. High values of degrees indicate a large number of relationships to entities. The rationale behind using this feature is that if a node is connected to a large number of entities, the likelihood is that it has been identified as performing many malicious actions over the Internet. Formally, the maximum degree of a vertex can be defined by $deg(v) = n - 1 \; \forall \; v \in G$.

Eigenvector centrality is the measure of influence a node has in a graph. A high eigenvector centrality value means that the node in question is connected to many nodes who themselves have a large number of connections. This feature is used to understand how important a particular node is within the graph as it gives a clear indication of how connected a node is, both directly and indirectly.

For the graph, G, let $A = (a_{v,t})$ be the adjacency matrix where

$$a_{v,t} = \begin{cases} 1 \; if \; vertex \; v \; is \; linked \; to \; vertex \; t \\ 0 \; if \; vertex \; v \; is \; not \; linked \; to \; vertex \; t \end{cases} \tag{5.2}$$

The eigenvector centrality score for a given vertex, v, can then be given as

$$x_v = \frac{1}{\lambda} \sum_{t \in M(v)} a_{v,t} x_t \tag{5.3}$$

where $M(v)$ is the set of neighbours of vertex v and λ is a constant. By virtue of the Perron-Frobenius theorum, the greatest eigenvalue, selected as λ, results in the desired centrality measure.

These features are then input to the k-means clustering algorithm offered by scikit-learn. The k-means algorithm is an unsupervised method, which partitions observations into k clusters, in which each observation belongs to a cluster with the nearest centroid. The k-means algorithm has a number of parameters, which can be configured to influence the result. The parameters chosen in this thesis are listed in Table 5.5. The n_clusters variable is changed to find the value which performs the best, and is discussed later in this evaluation.

The corresponding clusters within the graph can be used to identify nodes which are the most highly connected to the entities derived through CTI collection. Regardless of the number of clusters identified, there will also always exist a cluster which contains nodes which are the most loosely-connected. Nodes which are *not* part of this loosely-connected cluster include potential attacker nodes, which have a very high number of connections to blocklists and malware samples on the date the telemetry was captured, and as such they can be treated as supernodes. Any node in the graph which is not linked to a supernode is labelled

Parameter Name	Value	Description
n_clusters	Various	The number of clusters to form.
max_iter	300	Maximum number of iterations of the k-means algorithm for a single run.
n_init	10	Number of time the k-means algorithm will be run with different centroid seeds.

Table 5.5: The parameters configured for the k-means algorithm

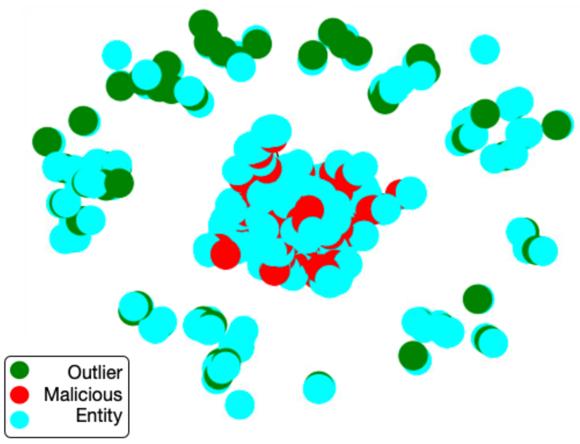


Figure 5.8: Graphical representation of nodes distinguished by label

as an outlier. These outliers can be further examined to deduce the intent behind their communication with a honeypot. The remaining nodes within the graph that are linked to a supernode, i.e. there exists an edge, or a series of edges, between the two nodes, are labelled as malicious. This is due to the fact that they share a common entity association. To find every vertex which is linked to a supernode, a breadth first search is performed. This procedure has a time complexity of $O(|V| + |E|)$ for each vertex, since all vertices must be explored in the worst case.

This approach forms the basis of the labelling of honeypot telemetry incorporated within LUFlow '20. Figure 5.8 presents a graph of all identified node relationships for a given date as derived through correlation with CTI services. The nodes which represent IP addresses extracted from the telemetry are coloured based upon the aforementioned labelling approach. The nodes labelled as malicious in red, as they are connected to a supernode, and outlier nodes in green. The cyan nodes indicate an entity as derived through CTI. The blue nodes within Figure 5.8 are the entities extracted through correlation with CTI services, such as blocklists and ASs.

Validation of the consistency within these clusters of data is required to ensure the identified supernodes, and derived ground truth, are accurate. The silhouette metric is used for this purpose. The silhouette metric shows how similar an object is to its own cluster, compared to other clusters. Silhouette measurements range from -1 to +1, where a high value indicates that the object is very similar to objects in its own cluster, and not similar to objects in other clusters. For any data point i within the cluster C_i let

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j) \tag{5.4}$$

be the mean intra-cluster distance, where $d(i,j)$ is the distance between i and j in the cluster C_i. The mean dissimilarity between i and a cluster, C_k, in which i is not a member can be defined by

$$b(i) = \max_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_i, i \neq j} d(i,j) \tag{5.5}$$

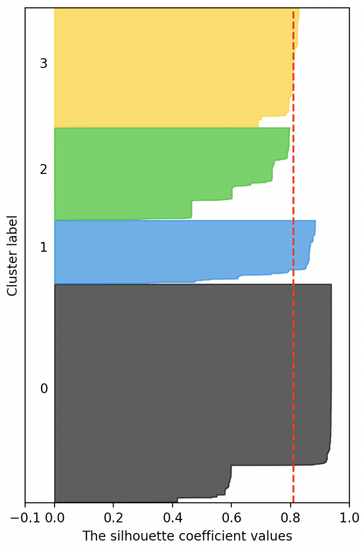
The silhouette coefficient of i can now be defined by

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, if\ |C_i| > 1 \tag{5.6}$$

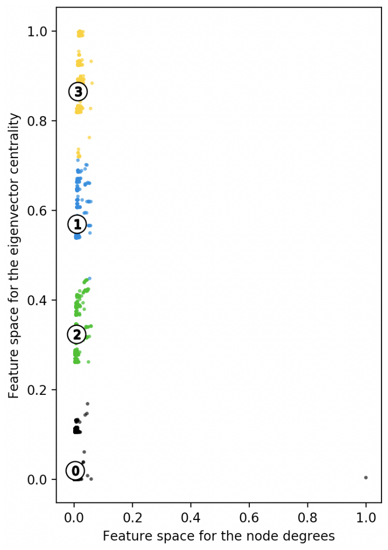
These measurements for each data point, i, can be displayed visually by combining the silhouettes into a single plot, enabling an appreciation of the overall quality of the clusters. Therefore, the average silhouette value, i.e. width of the plot, provides an evaluation of clustering validity [166].

5.3.2 Results

An example plot is presented in Figure 5.9(a), which shows silhouette values for various clusters of graph based features used to derive a ground truth. This example is taken from the results of clustering LUFlow '20 data captured on the 7th September 2020. Furthermore, this example is used to showcase the excellent



(a) Combined silhouette values which are distinguished by cluster.



(b) Clusters plotted in feature space.

Figure 5.9: Clusters visualised by silhouette values and feature space.

clustering capabilities provided by the k-means algorithm. In this case, it is visible that the clusters are dense, well separated and consistent with each other. This is because the clusters are of similar thickness, indicating a similar sample size, and contain high silhouette values, which are all in the region of the silhouette average, indicated by the vertical red dashed line. It should be noted that the green cluster contains data points which are less similar than others, as evidenced by the fluctuations in silhouette values. This is further visualised in Figure 5.9(b), which shows the clusters more clearly by plotting the corresponding data points in feature space. The different colours within the figure represent a cluster of data. Each of these clusters are identified using the aforementioned k-means algorithm. The colour of the cluster is linked to silhouette value with the same colour in Figure 5.9(a).

In order to fully evaluate how similar every supernode and outlier clusters are to each other, the average silhouette value is calculated for every graph used to label telemetry relating to LUFlow '20. As previously discussed, a graph is created for every date telemetry is captured to compile LUFlow '20. A time series of these values, ranging from June to December 2020, is presented in Figure 5.10, which displays an average silhouette value for variable cluster size, n.

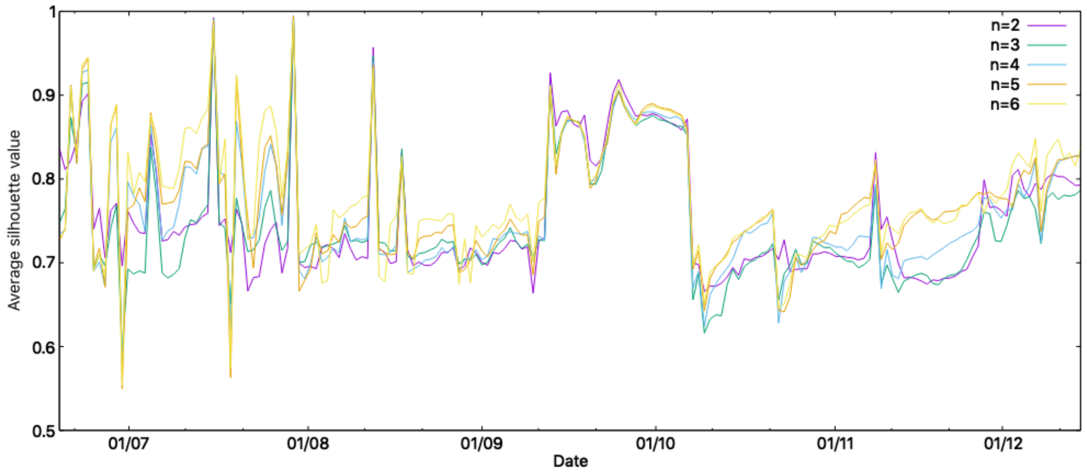


Figure 5.10: Average silhouette value time series

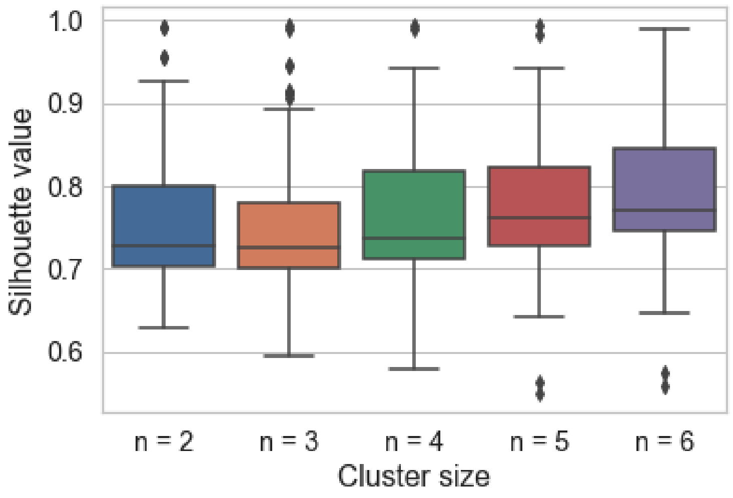


Figure 5.11: Box plot depicting distribution of silhouette values for varying cluster sizes.

As shown, the lowest average silhouette value is greater than 0.55, indicating clear cluster consistency in general. Figure 5.11 illustrates a box plot representing the distribution of silhouette values for varying number of clusters. Evidently, we can deduce that significant differences exist between the distribution of silhouette values based upon the size of the clusters chosen. The mean silhouette value for a cluster size of six is $\mu_6 = 0.7868$, and the standard deviation is $\sigma_6 = 0.0727$. When the cluster size is at the lowest, a value of two, the mean silhouette value is $\mu_2 = 0.7572$, and the standard deviation is $\sigma_2 = 0.0731$. Hence, it can be concluded that nodes within the supernode and outlier clusters truly belong in

Name	Labelling Mechanism	Open Source	External Correlation	Data Set Released	Attack Types
Citrus	Cyber Threat Intelligence	Yes	Yes	Yes	Various
MAWI [114]	Unsupervised AD	No	No	Yes	Various
Sperotto et al. [113]	Log Correlation	No	No	No	Various
Aparicio-Navarro et al. [115]	Dempster-Shafer	No	No	No	Wireless
B-IDS [117]	SVM, RPCL, and Dempster-Shafer	No	No	No	Various

Table 5.6: Comparison of Citrus and other frameworks which develop a ground truth.

those clusters due to the absence of any negative silhouette metrics.

The above results indicate that a cluster size of 6 is optimal. However, this is only for the data used in the analysis. As mentioned previously, this analysis uses data from June 2020 to Decemeber 2020. Since LUFlow '20 receives updates, this value is liable to change depending on the range of data used. It is recommended to perform this evaluation to identify the optimal value for data captured on different dates.

5.3.3 Comparison

This section compares Citrus' ground truth development capabilities through experimentation and a comparison of features. To begin, Table 5.6 provides a comparison between the features of Citrus and other similar frameworks. Citrus is shown here to be the only ground truth development solution which provides an open-source codebase within literature. Furthermore, Citrus is also the sole framework which performs external correlation for a greater understanding of potential intrusion attempts. The majority of the other solutions also neglect to publicly release the associated data set. Naturally, this does not aid further research efforts in the evaluation of network defense solutions using diverse attack data.

Citrus establishes data ground truth through an unsupervised clustering approach. In contrast to supervised classifications methods, which have been traditionally used in a multitude of systems (e.g., [91]), there is no requirement of costly training procedures. In order to demonstrate Citrus' advantages, an experimental comparison is performed between Citrus and a similar data set labelling framework, B-IDS. This framework was chosen as it leverages a number of open-source algorithms, which promotes the wider replication of results in this experimental comparison. These open-source algorithms are used to engineer B-IDS using the specifications provided in [117].

A single day of telemetry capture, comprised of one million records, is used in this evaluation to compare the approaches. This telemetry is labelled using the unsupervised and supervised approaches on the same hardware. The supervised B-IDS method was implemented using open-source software and was configured according to the specifications provided in [117].

Name	F-score	Recall	Accuracy	Precision
Citrus	95.37%	93.40%	93.87%	97.43%
B-IDS	94.30%	92.12%	93.51%	96.60%

Table 5.7: Mean classification metrics for attack scenarios using data set output by Citrus and B-IDS

Classification metrics were calculated for the online detection scenarios, outlined in Section 5.4.2.1, using the telemetry labelled by both Citrus and B-IDS. As evidenced in Table 5.7, the results show that Citrus outperforms the supervised alternative, and has an increased F-score metric by 1%. Due to F-score representing the harmonic mean between recall and precision, it can be concluded that Citrus' labelling methodology is superior in regards to enhancing detection mechanisms. In addition, the other classification metrics are marginally greater for Citrus, indicating overall superiority.

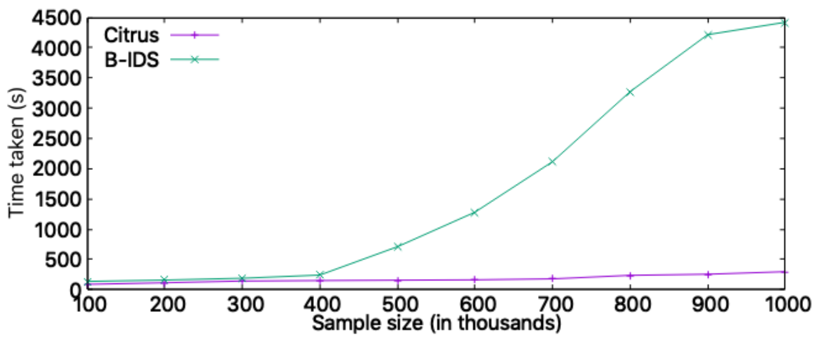


Figure 5.12: Comparison of computational cost to label varying number of samples.

Furthermore, the experimentation has also demonstrated that Citrus has a more optimal computational cost when compared to the alternative. Figure 5.12 illustrates the time taken to label a number of samples using both approaches. As shown, there exists similar computational cost for the lower number of samples.

However, when considering larger sample size, Citrus performs much better, indicating clear supremacy in this regard, and further supports its real-time detection pipeline.

5.4 Detection Capabilities

This section explores the detection capabilities enabled by LUFlow '20. This is demonstrated through leveraging the proof-of-concept implementation, Clementine. As previously discussed, Clementine performs intrusion detection by taking a machine learning approach. Two experiments are conducted which evaluate Clementine's capacity to detect a variety of emerging attacks. In the first experiment, an offline detection approach is taken, which considers various classification methods. In the second experiment, online intrusion detection is performed using live network data, which incorporates injected attacks, and the classification method which performed the best in the first experiment. The online evaluation is performed to assess Clementine's effectiveness in malicious behaviour detection using a realistic network environment.

In both of these experiments, Clementine is installed on a VM, which is provisioned with 4 vCPU cores and 32GB RAM, in an isolated network within the Cyber Threat Lab. Clementine also integrates with Spark to perform data processing and classification algorithms in parallel on the aforementioned cluster. The detection performance of each of these algorithms are assessed by comparing the predicted label to the actual label. This assessment forms a confusion matrix, which describes all possible classification outcomes. An example is shown below in Table 5.8.

		Actual Label	
		Benign	Malicious
Predicted Label	Benign	TN	FN
	Malicious	FP	TP

Table 5.8: Example confusion matrix

For each experiment within this evaluation, the flows labelled as outliers are removed from the data set, enabling a binary classification outcome. This is because the true intent of the flow is not known. In the experiments performed in this evaluation, a positive prediction is where Clementine produces a label of *malicious*. Therefore, a True Positive (TP) refers to a prediction of *malicious* when in fact the flow in question is *malicious*, else it is treated as a False Positive (FP). On the contrary, a negative result occurs when Clementine predicts a *benign* label. As a result, if the flow under scrutiny is not related to attack traffic and Clementine predicts a *benign* label, a True Negative (TN) occurs. Furthermore, if Clementine predicts a *benign* label when the flow is related to malicious behaviour, a False Negative (FN) occurs. From this confusion matrix, conventional metrics can be derived, which further assess the performance of the detection approach. These are outlined below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$
$$Fscore = 2 * (\frac{Precision * Recall}{Precision + Recall})$$

The accuracy of an algorithm represents the probability that a flow record is correctly classified. Recall indicates how successful the algorithm performs in identifying the positive class, while precision provides the percentage of positively classified flows which are truly positive. F score represents the harmonic mean of precision and recall.

5.4.1 Offline Detection

5.4.1.1 Methodology

This experiment evaluates the capacity of machine learning algorithms to detect malicious activity within LUFlow '20. A myriad of traditional supervised learning algorithms are considered and evaluated within this section. This experiment is intended to verify the validity of the data set whilst identifying the greatest performing algorithm to use in the online detection process. Motivated by similar research within literature [91], this experiment was also selected to validate the use of machine learning in intrusion detection scenarios. Despite this test not considering streaming data from within the network, it is still classifying real network telemetry captured previously. Thus, if a positive result is obtained then the test indicates that machine learning is a suitable detection approach for this type of data.

In this experiment, the LUFlow '20 data set is split into a training and test data set. The training data set consists of 10,000,000 randomly sampled records in total, with 5,000,000 normal records and 5,000,000 malicious records. The malicious records are the result of the telemetry captured and labelled by Tangerine, and incorporate a wide variety of unknown attacks. The test data set contains 8,000,000 records, split into 4,000,000 malicious and 4,000,000 benign classes. The number of records within each category was chosen based upon positive results in related research [91], while also maximising the number of flows captured within the data set. Ensuring a large number of flows are used ensures that many diverse attack patterns are included in the training data.

Algorithm	Parameter Name	Value
Logistic Regression	iterations	250
	regParam	0.01
	regType	l2
	intercept	False
	corrections	10
Gradient Boosted Tree	impurity	variance
	lossType	logistic
	maxDepth	5
	maxIter	100
Random Forest	impurity	gini
	numTrees	100
	maxDepth	5
	maxBins	32
	featureSubsetStrategy	sqrt
Naive Bayes	smoothing	1
Decision Tree	impurity	gini
	numTrees	100
	maxBins	32
Linear SVC	maxIter	250
	fitIntercept	True
	regParam	1
	tol	1e-06

Table 5.9: The parameters configured for various algorithms

5.4.1.2 Results

Training machine learning models enables inference and classification of future unknown records. Each of the machine learning algorithms associated with these models typically exposes tunable hyperparameters to developers, which adjusts how the algorithm performs internally. For example, the Radial Basis Function (RBF) kernel in SVM algorithms enables the specification of C and γ parameters. C acts as a regularisation parameter for SVM, while γ dictates the curvature of the decision boundary. It is not known beforehand which C and γ are optimal for a given problem, therefore, a *grid search* must be performed. The goal of this process is to identify the values for these parameters which increase prediction accuracy of unknown data. Table 5.9 documents the parameters used in each of the experiments.

The results of the offline detection approach is outlined in Figure 5.13. Each model listed within this figure has been trained with the parameters which maximise the aforementioned classification metrics. The optimal parameters for this classification problem were identified through a grid search of several values for each parameter used.

As shown, the worst performing supervised classification algorithm was Naive Bayes. This algorithm performed relatively well in recall, however, lacked in other metrics, suggesting that the algorithm is incorrectly labelling benign flows as malicious. On the contrary, one of the highest performing algorithms was the random forest classifier, which obtained 98.8% precision, as well as high accuracy, recall and F-score. Random forests belong to the class of ensemble learning algorithms, and can be used in both regression and classification tasks. They operate through the construction of numerous decision trees during training. Each individual tree is used to predict the class a data instance belongs to under classification scenarios, and the mode of these classes is used as the model's prediction. As well as the promising classification metrics, this algorithm was also identified to be faster than the comparable gradient-boosted tree. As a result, it is more appropriate for use in online evaluation. This is discussed further in the next section.

Despite LUFlow '20 being publicly released a short time ago, there has been research that evaluates it from a ML approach. Chua et al. [167] devise an experimental framework that performs classification of network data using ML.

Model Name	Recall	Accuracy	Precision	F-score
Logistic Regression	90.23%	87.64%	91.52%	90.87%
Gradient Boosted Tree	94.59%	95.17%	99.18%	96.83%
Random Forest	93.28%	94.27%	98.84%	95.97%
Naive Bayes	86.76%	38.12%	2.34%	4.56%
Decision Tree	91.04%	92.98%	91.52%	91.28%
Linear SVC	64.52%	64.23%	99.02%	78.13%

Table 5.10: Offline classification metrics for a range of ML models

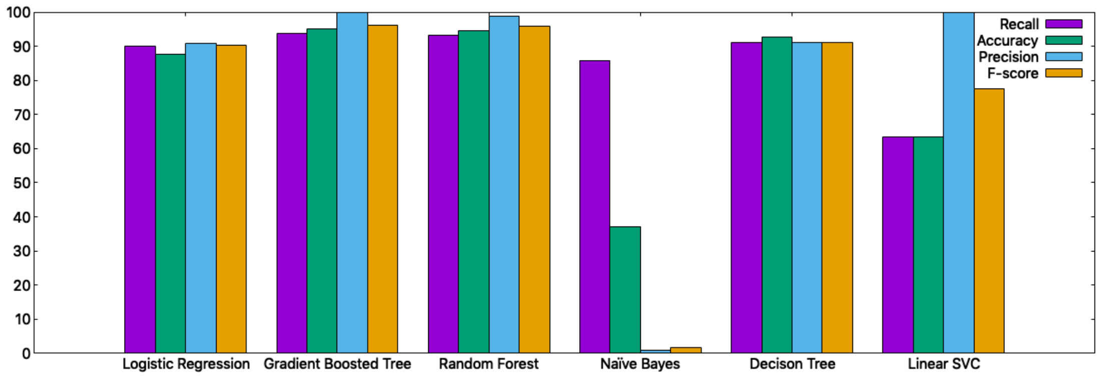


Figure 5.13: Statistical metrics observed from offline detection

They select two data sets for this evaluation: CICIDS2017 and LUFlow '20. In their work, they identify similar classification metrics to the results in this evaluation. For example, Naive Bayes performs less well than the other classification algorithms and Random Forest performs the best. With the exception of Naive Bayes, all of the algorithms used in their work resulted in accuracy of over 99%. These results are much better than the results found in this evaluation despite using the same algorithms and data set.

To understand why this is case, one must identify the samples of data used. This evaluation uses randomly sampled data from June 2020 to December 2020 for training and test data, while Chua et al. use June 2020 as training data and January 2021 as test data. They also sample only 20% of data from these months to further reduce the number of flows under evaluation. The sample size in their work is much lower when compared to this evaluation, thus, the number of attacks they need to correctly identify is also lower. Furthermore, this also highlights the benefits granted by training on smaller and more relevant samples, driving future efforts into the automated iterative training on recent data.

Chua et al. also compare the results of LUFlow '20 to CICIDS2017. Interestingly, they observe different results from the data sets. It was identified that models trained from CICIDS2017 suffered from different degrees of overfitting. This is in contrast to models trained on LUFlow '20, where no overfitting was observed. This could be attributed to the differences in the attacks and environment for CICIDS2017. CICIDS2017 changes various aspects about the environment, including servers where data is captured, while LUFlow '20 has a stable environment where the attacks change over time.

5.4.1.3 Feature Importance

For the detection experiments listed within this thesis, all features incorporated within LUFlow '20 are used. The random forest model fit in this experiment also grants an insight into the importance of each feature. The feature importance is denoted by a weighting as identified by the random forest model. In this instance, gini importance is used by random forest to identify these values. Each of these weights specify how critical a certain feature is in the determination of benign or malicious intent. The higher the value, the more critical the feature. The top five most important features within LUFlow '20 are documented within Table 5.11.

Feature Name	Weight
bytes_out	0.422
total_entropy	0.205
duration	0.100
entropy	0.072
num_pkts_in	0.072

Table 5.11: The importance of each feature

As shown, the number of bytes transmitted from source to destination within a flow is regarded as the most important feature within the data set in regards to classification between normal and malicious activity. This suggests that there may exist evident differences between this value for benign and malicious instances.

Upon further analysis of network flows, it is identified that there does exist a notable difference between the value for bytes_out when comparing flows with distinct target labels. Figure 5.14 provides a box plot illustration of this comparison. As shown, there is an evident visual discrepancy. The mean value for

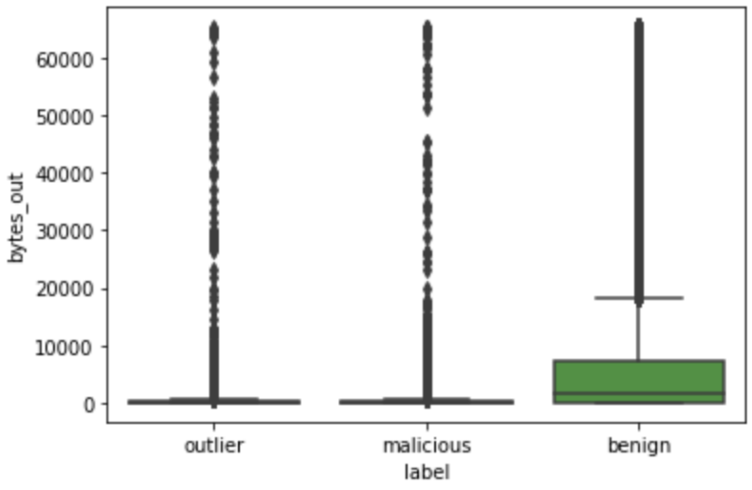


Figure 5.14: Statistical metrics observed from offline detection

bytes_out of benign flows is $\mu_{benign} = 4182.68744$, and the standard deviation is $\sigma_{benign} = 6292.4795$. The same values for malicious flow traces are much smaller, with $\mu_{malicious} = 331.3217$ and $\sigma_{malicious} = 1476.9448$. Through this analysis, the results within Table 5.11 can be considered as expected.

These statistical properties are a result of behavioural distinctions between traffic belonging to malicious and benign network traces. With an understanding of these differences from a practical perspective, the results documented can also be expected. In detail, a large portion of the benign behaviour corresponds to traffic relating to communication with a distributed database to store telemetry data. This communication is long-lived and large amounts of data are transferred within the connection. Attack traffic such as scanning contrasts heavily with these types of network flows. For example, TCP SYN scanning sends a single packet with SYN in the packet header.

Feature importance can play in important role in tuning classification algorithms. While we consider all features within this evaluation, altering the number of features can remove non-essential features and provide positive results. Chua et al. perform analysis on LUFlow '20 and identify such a relationship. In their research, they find that the Naive Bayes algorithm provides the lowest accuracy when one feature is considered, and it provides the highest accuracy when three features are used. Chua et al. also provide a list of the most important features identified within their research. In the same manner as this evaluation, they also use random forest to calculate the feature importance. They discover different results when compared to the feature importance documented in this evaluation. This could be due to the use of different data to perform the evaluation, as the authors only use data captured in June 2020. As identified in Section 5.2.4, the attack activity evolves over time, which could have an influence on the feature importance depending on the selected data period.

5.4.2 Online Detection

5.4.2.1 Methodology

This experiment leverages the random forest classifier to detect malicious behaviour in an online fashion using live network telemetry. The random forest classifier was carried over from the previous experiment as it resulted in high classification metrics and performed faster than the gradient-boosted tree model. The data used to train the model consisted of 10,000,000 malicious and benign records respectively. The parameters used for the random forest model are the same as the ones which were identified to perform the greatest in the offline detection evaluation. After having successfully trained the classifier on this training data, Clementine is ready to begin the consumption of live network telemetry in order to perform online intrusion detection. To achieve this, Clementine leverages Spark's Streaming abstraction. In this evaluation, Kafka is used as the medium in which telemetry is transferred from local network devices to Clementine, where it is then processed and classified into benign or malicious binary classes. Kafka was chosen as it provides a highly scalable message broker, capable of processing vast amounts of data with low latency. This is essential when considering that attacks need to be detected as soon as possible, and within large scale networks.

For the purpose of this experiment, additional VMs are provisioned within the Cyber Threat Lab. These are scattered throughout the various networks contained within. An illustration of how these are connected within the network to detect real attacks is included in Figure 5.15. A victim server is instantiated within the shared infrastructure to allow intrusion attempts from attack servers located in the other networks. The victim server performs similar functionality to the benign network services used to profile normal behaviour within LUFlow '20, including coordination with a distributed database amongst other known normal behaviour. A variety of victim machines were chosen to evaluate the detection properties across a range of system implementations. The attack servers attempt

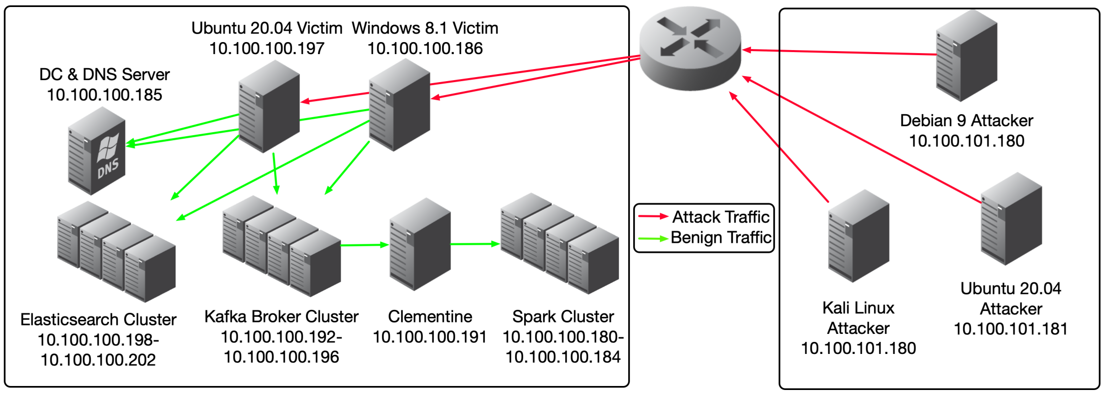


Figure 5.15: Configuration of experimental set-up.

to infiltrate the victim server through various means. These infiltration attempts are identified as being representative of emerging attacks currently propagating through the analysis of the telemetry captured by honeypots, as well as related research and reports [27, 168].

The victim server's network telemetry is captured by the aforementioned Joy tool. This is then immediately transferred to a Kafka broker where it is then consumed by Clementine. As this telemetry is generated irrespective of Tangerine, the labelling process is not the same. Due to the prior knowledge about the injected attacks, flows are labelled based upon the IP addresses associated with the flow. If either the destination or source IP address correspond to an attack server, the flow is labelled as malicious, otherwise benign.

In this evaluation, a series of separate experiments are conducted, which examine Clementine's ability to detect emerging threats in an online fashion using live network telemetry. These experiments are conducted to evaluate the accuracy from a practical perspective. Similarly to the offline evaluation, this experiment validates ML based detection capabilities on network data. However, this experiment also considers live data. Thus, it also examines the capabilities within a real networked environment. Such an environment requires methods to capture and transmit the data from diverse devices in near-real time, which could impact the result when compared to offline detection. This evaluation also considers previously unobserved attacks and attacks on systems other than the honeypots where the training data was gathered. These properties were selected to understand how effective the detection of an unsuspected attack is within an organisation's network, which consists of heterogeneous systems. Network telemetry is captured and streamed to Clementine over a period of 10 minutes. At a random point during this period, the normal telemetry is injected with malicious behaviour through attacks performed by various attack servers. The various scenarios used in this evaluation are documented below.

Scanning In this scenario, a single attacker host performs a port scan of the victim machine. The victim machine in this case is a standard Ubuntu 20.04 VM, which is instantiated in the Cyber Threat Lab. The Nmap tool is used to perform such network scan. Specifically, TCP SYN and UDP scanning techniques are leveraged to identify thousands of open ports upon the victim machine. These scanning attempts send multiple packets to the victim, which further compose the associated malicious flows streamed to Clementine through Kafka. After 10 minutes of normal and injected malicious behaviour, the experiment is stopped and the associated metrics are calculated.

DDoS This attack involves flooding the victim machine with a large number of TCP-SYN requests to overwhelm its resources. This is orchestrated by numerous attack machines located within the same network. These machines initiate flooding for a period of 2 minutes. This is achieved through leveraging the hping3 tool. The tool is configured with a random source address, flood rate, and SYN packets. The flood rate configuration sends packets as fast as possible and do not wait for a reply before sending another. Every benign and malicious flow is captured and streamed to Clementine, which classifies each in an online manner.

Brute Force The use of brute force and dictionary attacks plague networked systems to this day. Sending rapid bursts of authentication requests, these attacks can be orchestrated against remote or local targets to gain a foothold into infrastructure and gather sensitive information. In this experiment, an MSSQL server is targeted with a large amount of brute force traffic for a period of 2 minutes. Hydra is used to perform this attack, further utilising the largely popular 'rockyou.txt' wordlist [169] as the attempted passwords.

Exploitation In this experiment, an attack is leveraged against a victim machine, which consists of a number of distinct stages. Initially, the NSA exploit leaked by the Shadow Brokers collective, which takes advantage of CVE-2017-0145, EternalRomance, is utilised to gain remote code execution privileges on the victim machine. Despite public disclosure in 2017, this exploit still remains a prominent threat in recent times [170, 171]. This exploit delivers a payload, which is then executed. The payload is composed of shell code, which instantiates a reverse shell on the victim. This reverse shell provides a covert method of interaction with the victim, and is further used to perform various actions, such as downloading sensitive documents to model data exfiltration in this experiment. These separate stages are designed to emulate emerging attack patterns encountered in real-world scenarios.

As the EternalRomance exploit leverages Microsoft's implementation of the SMB protocol, a Windows 8.1 VM was instantiated within the Cyber Threat Lab to play the role of the victim. In the same vein as the other experiments, the joy tool is built to capture live telemetry in the form of network flows. All stages within this attack against the victim machine were orchestrated by a VM within the Cyber Threat Lab running a Kali Linux distribution [172]. A reverse shell payload is generated using the msfvenom framework, specifying the target architecture and payload format. The meterpreter reverse shell handler [173] is then used to listen for an incoming connection from the victim. Next, using the Metasploit framework installed on Kali, the EternalRomance exploit is chosen, with further configurations including setting the destination host to target the Windows victim machine and the payload to the reverse shell. The exploit is then executed against the victim, gaining access to the aforementioned reverse shell through execution of the post-exploitation payload.

Cryptojacking As identified in Section 2.1.1.1, cryptocurrency mining is popular amongst cyber criminals as a means of generating large amounts of income. In order to perform mining, a server must first be infiltrated through various methods. Much like the previous experiment, this experiment also consists of multiple stages. In order to initially compromise the server, the SSH credentials of the victim machine are found through a dictionary attack using the aforementioned Hydra tool. A custom password list, containing the password of the victim server, is used by Hydra to identify the correct credentials.

Upon successfully cracking credentials, the Secure Copy (SCP) binary is used, alongside the credentials, to copy XMRig [174] software to the victim machine. XMRig is cryptocurrency mining software, which can be used for legitimate purposes, however, it is also used by cyber criminals to mine cryptocurrency on exploited servers [43]. A remote shell is then opened on the victim machine using the credentials harvested previously, and the XMRig software is executed.

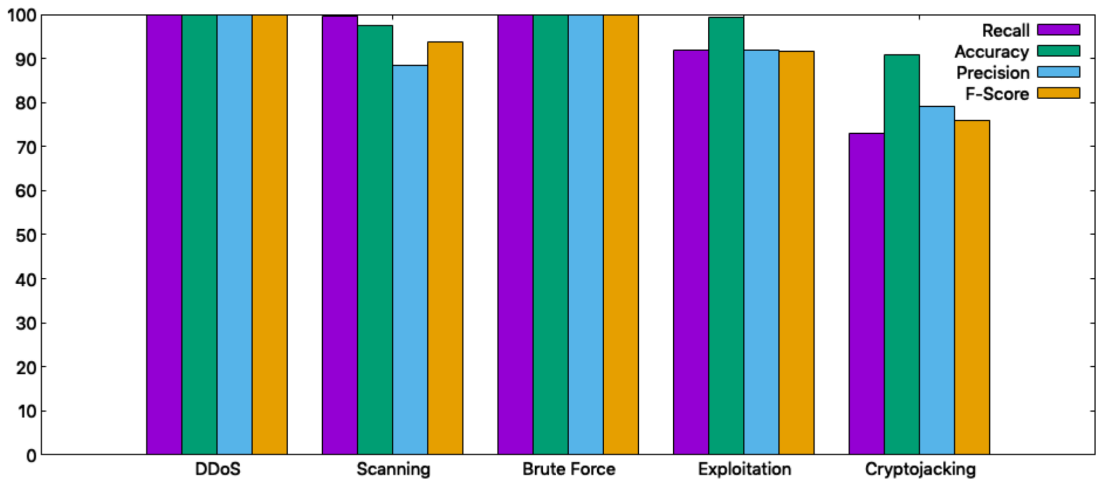


Figure 5.16: Performance metrics obtained from online classification of live telemetry

5.4.2.2 Results

Attack Type	Recall	Accuracy	Precision	F-score
DDoS	99.78%	99.79%	99.87%	99.82%
Scanning	99.78%	97.46%	88.43%	93.80%
Brute Force	99.78%	99.79%	99.76%	99.77%
Exploitation	91.43%	99.12%	92.65%	92.03%
Cryptojacking	71.57%	90.33%	79.08%	75.13%

Table 5.12: Online classification metrics for a number of attack scenarios

Scanning As shown in Figure 5.16, these metrics showcase Clementine's high accuracy, 97.46%, and extremely high recall, 99.70%. This demonstrates the ability of Clementine to detect scanning attempts using live telemetry. Despite the high recall, the precision in this experiment is markedly lower. This could be attributed to a difference in scanning methodology between data captured at honeypots and in this experiment. It is likely that the data captured by the honeypots represents horizontal scanning techniques since these IP addresses serve no legitimate purpose and can only be discovered through this method. Conversely, the scan type in this experiment uses a vertical method.

DDoS The performance metrics associated with the classification is illustrated in Figure 5.16. As shown, Clementine performs extremely well under a DDoS attack; demonstrating a low number of FPs and FNs and over 99% in all calculated metrics. This attack is orchestrated through the rapid transmission of SYN packets. Such an attack involves network flows which exhibit similar properties. Furthermore, this attack does not require high-interaction honeypots to capture the most realistic data since no underlying service is required to be exploited. These types of flows should be easily distinguished from benign behaviour, therefore, detected with high accuracy.

As discussed in the related work, Sangkatsanee et al. also perform a similar evaluation using a private data set. In their work, they perform online classification using live network traces, which include DDoS traffic. The results are similar to those found in this evaluation with over 99% accuracy in the classification of DDoS attacks.

Bruteforce As shown in Figure 5.16, Clementine is able to successfully distinguish between benign flows and flows relating to brute force attempts with extremely high precision, accuracy, and, recall. In a similar manner to the DDoS experiment, this attack does not require high-interaction honeypots to provide realistic training data. As a result, if bruteforce attacks were captured within the Cyber Threat Lab the training data would contain very similar network flows to those found in this experiment. Thus, the ML algorithm can accurately identify malicious activity.

Exploitation As illustrated in Figure 5.16, Clementine classifies network flows under this attack scenario with very high accuracy, over 99%, with all other metrics achieving above 90%. Critically, flows associated with the meterpreter reverse shell are correctly classified as malicious. As previously discussed, the malicious telemetry captured to create LUFlow '20 emanates from medium-interaction honeypots. Therefore, this specific attack does *not* exist in training data as payloads are captured but not executed, and can be treated as a novel attack vector. The ability of machine learning algorithms to detect unknown attacks is clearly highlighted in this scenario.

Furthermore, within this scenario, Clementine only incorrectly classified a single malicious flow record as benign. This is the first truly malicious flow record involved in this multi-stage intrusion experiment, and includes the initial exploitation attempts. Due to the semantic gap inherent within ML algorithms, there can be no explicit explanation, however, there exist a number of possibilities. The first potential reason is that the shell code payload delivered through the Metasploit framework is substantially different to that of the WannaCry worm

(3611 and 4869 bytes in size respectively [175]). The WannaCry worm is reported to remain a prominent threat [176], and upon manual inspection over 90% of the binaries captured from the Dionaea honeypots within the Cyber Threat Lab are attributed to WannaCry according to VirusTotal [177]. Another possible explanation for this occurrence is the fact that the emulation of SMB services offered by Dionaea is not robust enough. As identified in issues posted by the developers, the emulation is currently not perfect and leaves a lot to be desired [175]. It would be an area of interesting research to repeat the experiment with data captured by legitimate SMB service and compare the results. Despite this single misclassified malicious flow, the remainder of malicious flows are correctly identified, ensuring that the attack is still successfully detected in a timely manner.

Cryptojacking Crytocurrency mining is performed on the victim machine for a period of 5 minutes. As illustrated in Figure 5.16, Clementine classifies all stages within this multi-stage attack with 90% accuracy. The remaining metrics associated with this attack are slightly lower. This is in part due to a number of False Negatives (FNs) associated with the shell session opened by the attacker to the victim machine. Clementine classifies some instances of flows relating to the shell as benign, despite it being used for malicious purposes. While this is the case, Clementine is able to accurately classify the dictionary attack and cryptocurrency mining stages performed by the attacker machine. These stages within the attack are undeniably malicious as they are not part of the normal telemetry profile within LUFlow '20. As a result, detection of this attack as a whole is enabled by Clementine. In addition, due to the nature of the network environment used to capture traffic which composes LUFlow '20, cryptocurrency mining traffic is not contained within the training data used within this experiment. As a result, the cryptojacking stage within the attack can be considered a previously unobserved attack, thus, further highlighting the benefits of an anomaly detection approach to intrusion detection. In the same vein as the exploitation experiment, due to stages within this attack not being observed before in training data, this type of attack could exhibit different statistical properties to known malicious behaviour. Therefore, the results observed could be impacted accordingly.

5.5 Classification Performance

A performance evaluation is also performed which considers the efficiency with which Citrus processes data. This section examines a variety of Clementine's performance aspects to understand the benefits gained through leveraging the Spark parallel processing engine.

5.5.1 Model Training

5.5.1.1 Methodology

In typical embodiments of grid search functionality, a separate model is trained *sequentially* on a single server for each parameter value combination specified in the grid search. The best performing model, i.e. the most accurate, is then selected for future prediction purposes. The Spark engine provides an alternative implementation, which leverages the power of cluster computation. In this approach, separate models can be trained in parallel across a number of executor nodes in the cluster. This has the benefit of combining heterogeneous computational resources to tackle computationally expensive problems.

In order to evaluate the performance benefits of this approach, several experiments are conducted, which aim to compare grid search implementations and the corresponding overall training time of machine learning models. Based upon the favourable detection results obtained in Section 5.4, the Random Forest classifier is used within these experiments. In each, the model is trained with 500,000 randomly sampled records extracted from the LUFlow '20 data set. To compare the distributed training approach to a baseline, the widely popular sklearn library [153] is leveraged to perform parameter tuning and model training in a single core scenario. This library provides the necessary random forest and grid search implementations. Both Spark and sklearn share parameters used in their respective random forest classifier implementations. These parameters can be modified to influence the outcome of the decision making process. The various parameters and values used in these experiments to evaluate and compare the efficiency of distributed model training against a single core approach are included in Table 5.13.

In order to measure the total time taken to build several models in parallel

Parameter Name	Values	Description
numTrees	10, 20, 40, 80	The number of trees within the forest
maxDepth	1, 3, 5	The maximum depth of the tree
impurtiy	entropy, gini	Criterion used to evaluate quality of a split

Table 5.13: The parameters used to evaluate single core and cluster approaches to model training

using Spark, the package SparkMeasure[1] is used. This package simplifies the collection and analysis of Spark performance metrics. This package is implemented using listeners to collect task metric data, and provides developers an API to accurately measure the time taken to perform various operations on a cluster.

Measurements are recorded for the total time taken to train all models, as specified by the parameters, and find the best performing upon a cluster of varying node sizes. In total, 4 Spark executor nodes, each containing 4 processor cores, are used to provide a maximum of 16 cores. The number of nodes in the cluster is used as an independent variable, and is changed to determine how it affects the total computation time. The mean of five separate measurements for each experiment is taken to record an average total training time.

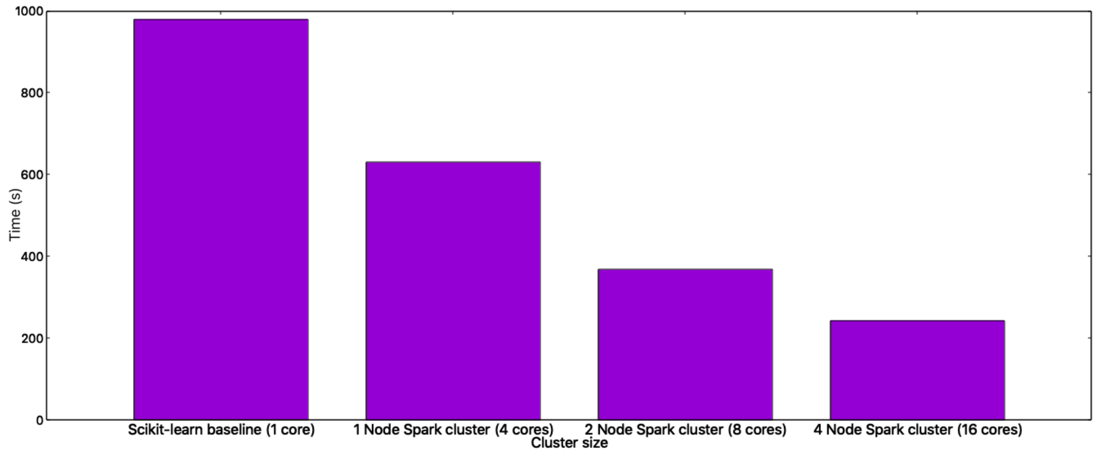


Figure 5.17: Comparison of model selection using grid search and variable cluster size

[1]https://github.com/LucaCanali/sparkMeasure

5.5.1.2 Results

As illustrated in Figure 5.17, sklearn's grid search implementation takes the longest time, 979 seconds, to find the optimal parameter combination. As previously discussed, this is because each model is trained sequentially. The parallel processing capabilities Spark provides are evident even on a single node cluster. Spark is able to leverage the 4 cores on a single machine, training models independently and reducing the total time taken compared to traditional approaches. This reduction in model training and selection is further evidenced in clusters of larger node sizes. Within a 4 node cluster, the average computation time is 243 seconds, a 75% reduction when compared to the baseline.

These experiments clearly demonstrate the performance benefits gained from leveraging Spark and a cluster of nodes, when compared to a single server approach, to train and search for optimal model parameters. Furthermore, the results have shown that increasing the number of nodes within the cluster, and associated cores, has a favourable effect on the overall training time. This is directly influenced by the increased level of parallelism used in processing operations.

5.5.2 Online Prediction

5.5.2.1 Methodology

The time taken to train a model measures the efficiency in which a behaviour profile is built, which does not fully evaluate the properties required for near real-time detection. To evaluate Citrus' capacity to detect threats in a timely manner, this section documents experiments which assess the performance of live data prediction. These experiments consider the temporal efficiency of all stages within the data classification pipeline, including data pre-processing and prediction. Notably, since Spark performs lazy evaluations of transformations, including prediction of telemetry, an additional action must be performed after each transformation to evaluate the processing efficiency.

As previously discussed, the Clementine module within Citrus integrates with Spark's DStream abstraction and performs intrusion detection in an online, practical fashion to detect emerging threats. DStreams enable the receipt of live input data streams and divide the data into batches, where it is then processed. Apache

Kafka is used in this embodiment to stream telemetry from heterogeneous VMs within the Cyber Threat Lab. The time taken to process and classify unknown records is of great importance, and should be as little as possible to rapidly notify systems administrators that malicious actions have taken place.

When compared to signature-based approaches, machine learning algorithms exhibit greater processing overhead. However, they contain powerful detection properties, capable of defending against unknown attacks. This evaluation aims to assess the speed at which a large amount of data can be streamed, processed, and classified using the proof-of-concept implementation, Clementine. As previously discussed, DStreams separate input data into batches. The interval of these batches determine the rate at which data is processed. Therefore, careful consideration should be taken when choosing this value. The batch interval is used as an independent variable in this evaluation to determine an optimal value in this instance.

Data containing 100,000 randomly sampled records extracted from LUFlow '20 is streamed to Clementine, where it is then pre-processed and classified, using the random forest classifier, into malicious or benign classes. Five separate measurements are taken for every batch interval value to derive the average time taken to process all streamed records. Importantly, the values reported in the following section pertain to the overall processing time taken to classify 100,000 records, and does not include any time waiting, as dictated by the batch interval, for the batches to fill with telemetry.

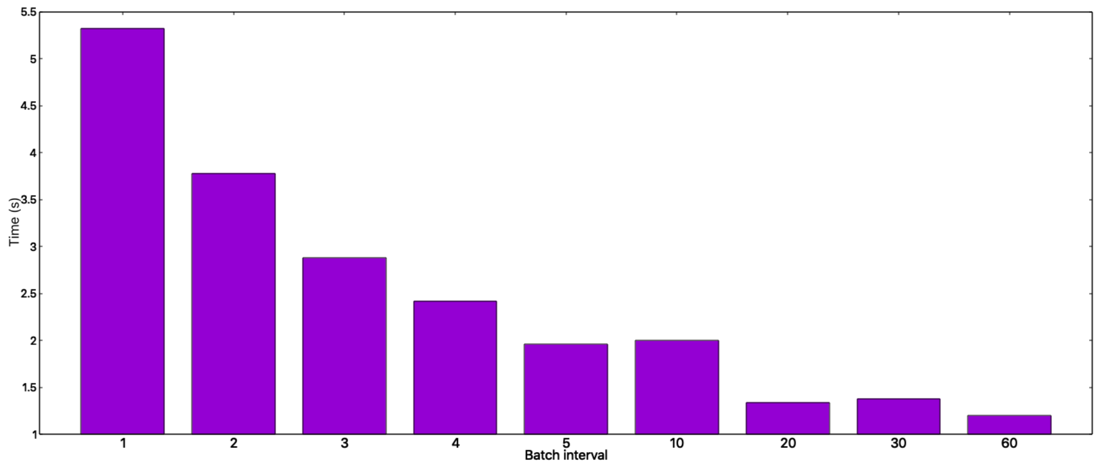


Figure 5.18: Evaluation of classification efficiency using variable batch interval

5.5.2.2 Results

As illustrated in Figure 5.18, the batch interval value has a significant impact upon the total processing time. In general, as the batch interval increases, the processing time decreases. For example, specifying a batch interval of 1 second separates the data into multiple mini-batches, all of which incur scheduling overheads. On average, the 1 second batch interval took $\mu_1 = 5.32$ seconds to classify all 100,000 records. In comparison, a batch interval of 60 seconds took an average of $\mu_{60} = 1.2$ seconds. This is because, in this case, all of the input data can be streamed within the batch interval, ensuring only a single batch to be processed. However, this interval means at least 60 seconds transpire before any classification occurs, and therefore does not allow near real-time prediction.

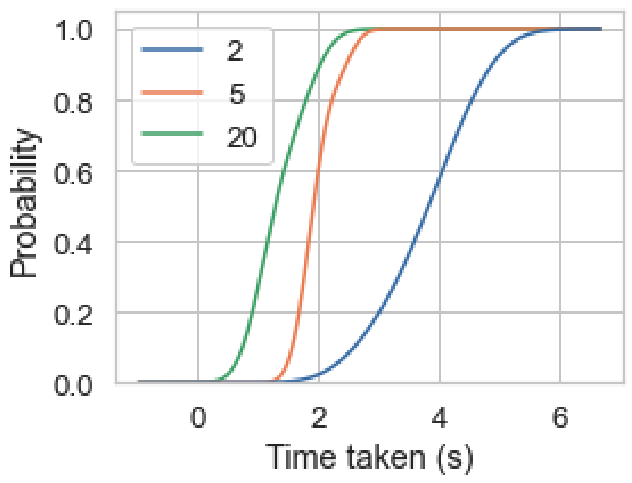


Figure 5.19: Cumulative Distribution Function (CDF) for processing time distinguished by batch interval.

In order to further investigate the distribution of processing times, additional analysis is performed on these measurements. Figure 5.19 displays a Cumulative Distribution Function (CDF) plot for measurements taken with a batch interval of 2, 5, and 20 seconds. Notably, there exist differences between the distribution of the measurements with respect to the batch intervals. The mean processing time with a batch interval of 2 is $\mu_2 = 3.780$ seconds, and has a standard deviation of $\sigma_2 = 0.691$. When the batch interval is 5 seconds, the mean processing time is $\mu_5 = 1.96$, and has a standard deviation of $\sigma_5 = 0.268$. The mean processing time for an interval of 20 is $\mu_{20} = 1.34$, and has a standard deviation of $\sigma_{20} = 0.389$. Despite the large difference in interval times, it is apparent that intervals of 5 and 20 seconds exhibit similar statistical properties. In this instance, an interval of 5 seconds is recommended due to the lower waiting time.

In order to understand the rate at which Clementine can process flows per second, another experiment is also performed, which leverages a batch interval of one second. Through five distinct runs of this experiment, an average of almost 11,000 flows were observed to be streamed to Clementine from another device within the network within each batch. This result showcases Apache Kafka's ability to process and transfer large amounts of data over the network with low latency. In addition, these flows are required to be processed by Clementine to classify potentially malicious traffic. It took on average around 0.9 seconds to process all data within these batches where a one second interval is used. Given this, on average it takes around 17 milliseconds to classify a single flow using a 1 second batch interval. This should be considered a conservative assessment, since results have also shown that batches containing over 20,000 records can be efficiently processed in under a second. Importantly, in this scenario, since the processing time is less than the batch interval of one second, there is no scheduling delay in which Clementine must wait to process previous batches before beginning to process the latest batch.

These results are extremely positive since they demonstrate Citrus' ability to rapidly process large amounts of data, and pave the way towards the composition of real-time detection. Due to this, it can be concluded that Citrus is able to be leveraged for the detection of emerging threats in large-scale networked environments.

5.6 Summary

Within this chapter, Citrus is evaluated in numerous ways. Every evaluation explored within this chapter was designed to examine a certain facet of Citrus, with the overall intention to demonstrate its effectiveness as an intrusion detection framework. Furthermore, these evaluations are also intended to assess whether Citrus has achieved the design requirements discussed in Section 3.1. A discussion of the results obtained through these evaluations is included in the remainder of this section.

In order to detect attacks using a machine learning approach, realistic training data which incorporates diverse attack patterns is required. Section 5.2 presents the analysis of the LUFlow '20 data set produced by Citrus for this purpose. Critically, this data set includes real attack telemetry captured through the composition of diverse honeypots. The analysis explores the geographic distribution of attack sources, as well as the evolution of the most popular services which are actively being exploited. In addition, a comparison is made between LUFlow '20 and other similar data sets within literature. This showcases the many beneficial properties inherent within LUFlow '20, particularly with respect to the updated nature. The telemetry analysed within this evaluation has showcased the availability of diverse attack data composed by Citrus. From the analysis of the telemetry, coupled with the fact that it has been released publicly[2] and continues to receive updates, it can be concluded that Design Requirement 1 has been achieved. The analysis contained within this evaluation also documents the features engineered from the raw flow capture. These features which compose the LUFlow '20 data set are discussed further in Section 5.2.5, and as a result, Design Requirement 2 has been met.

Intrusion detection data sets must be labelled when utilised by supervised machine learning algorithms. A ground truth must be established for the labels to be accurate. The evaluation performed in Section 5.3 includes a validation of the ground truth established by Citrus to label the LUFlow '20 data set. This involves the examination of the consistency between clusters used to separate benign and attack traffic. The time-series analysis performed in this evaluation has highlighted the routinely positive silhouette values across all dates where telemetry has been captured and labelled. This has ensured that the clusters used to identify malicious nodes are generally consistent, and do not include nodes which are actually representative of benign behaviour. Fundamentally, this validation of the ground truth guarantees the accuracy of the labels provided in LUFlow '20, and further enhances anomaly detection algorithms with robust normal and malicious behaviour profiles. The consistency between clusters amongst all days wherein telemetry has been captured and labelled demonstrates Citrus' ability to successfully develop a robust ground truth through correlation with third-party CTI services, thus, Citrus has successfully accomplished Design Requirement 3.

Section 5.4 presents an examination of Citrus' ability to detect attacks. This

[2]https://github.com/ruzzzzz/LUFlow

is assessed through the calculation of recognised classification metrics, which establish how effective Citrus performs under diverse attack scenarios. Initially, experimentation is performed to identify the highest performing machine learning model using attacks incorporated within LUFlow '20. The chosen model is trained on randomly sampled data from LUFlow '20 and used by Citrus to further evaluate its detection performance in real-world practical settings. This evaluation has highlighted Citrus' remarkable ability to accurately detect various emerging attack patterns using live telemetry emanating from real devices within the network, establishing it as a viable network defense solution.

In order to satisfy the elevated data processing requirements inherent within modern networks, Citrus must provide an efficient solution to network intrusion detection. The final evaluation (presented in Section 5.5) explores the efficiency of Citrus' data pipeline to classify network flows. This is first assessed through an examination and comparison of model training and parameter tuning using a cluster of varying sizes. The evaluation demonstrates that an increased number of executor nodes within a cluster drastically reduces the overall training time.

In addition, experiments are conducted to evaluate the efficiency of the decision making process taken by Citrus to identify network attacks in live telemetry using the model trained previously. Citrus was shown here to be able to classify tens of thousands of streaming flows within mere seconds using a small cluster deployed on commodity hardware. This highlights the beneficial properties relating to Citrus' efficient data pipeline in the detection of attacks orchestrated against real devices within the network. The practical deployment and evaluation of Citrus within a networked environment has generated largely positive results. In addition to the detection capabilities discussed previously, Citrus has demonstrated the ability to process and predict vast amount of telemetry in a timely manner. Hence, Citrus has successfully managed to provide accurate and near real-time intrusion detection capabilities suitable for modern networks. Therefore, it can be concluded that Design Requirement 4 has been achieved.

Chapter 6

Conclusion and Future Work

The modern threat landscape consists of novel and innovative infection mechanisms orchestrated en masse to deal damage to networks at scale. In parallel, the complexity of modern network infrastructure renders traditional approaches to monitoring and attack detection redundant, due to the variety and volume of data flow contained within. The further integration of emerging network technologies into everyday life, such as 5G, IoT, and Starlink satellites facilitate the continuous growth in attack surfaces and the number of network connected devices. These challenges require a substantial overhaul to monitoring and detection approaches in order to deal with the growing threat of malicious behaviour encompassed within the vast amount of data produced in modern networks.

This thesis acknowledges the importance and implications of these issues, and presents a practical solution for next-generation network defense and resilience strategies. This is achieved through the examination of emerging technologies, which leverage a cluster of compute resources to drastically reduce processing costs when deployed on large data sets. The solution presented in this thesis utilises such novel technologies to tackle the growing scalability challenges associated with data collection and processing techniques within modern network infrastructure. Furthermore, this thesis explores approaches which are able to deliver near real-time classification, in stark contrast to offline batch processing, which provide timely and effective detection capabilities appropriate for contemporary network infrastructure.

When deploying supervised machine learning algorithms to classify malicious behaviour in an online practical setting, models must be trained beforehand using data appropriate for the network environment. These models can then be used in a live fashion to perform an online assessment of network traffic. This thesis also investigates publicly available data sets suitable for this purpose. As previously identified, the most popular data sets used in research are considered archaic and lack attack traffic which is truly representative of modern infection vectors.

A data set of high quality should also include accurate labels, which represent the true nature of the data. The labelling technique adopted by creators of data sets are varied, however, they are typically derived through labour intensive methods of manual analysis. Recently, developments within the security community have emerged, which automatically label data sets through the establishment of a ground truth. A ground truth explicitly separates the benign from malicious traces within a data set, further bolstering anomaly detection techniques with a robust normal behaviour profile.

6.1 Thesis Contributions

As discussed in the introductory chapter, the ultimate goal of this thesis is to answer various open research questions. To begin this process, a survey of literature was conducted, which resulted in the identification of a number of challenges within research. Amongst others, these challenges limit the utility of anomaly detection techniques which are appropriate for modern network infrastructure. These challenges, in unison with an understanding of existing systems and future network directions, have influenced the design of a novel intrusion detection framework: Citrus.

The design of Citrus is segmented between various modules, which are responsible for performing specific functions. In detail, the design of the Tangerine module lays out the necessary components required to collect and label telemetry to output a novel intrusion detection data set. To ensure the accuracy of the labels, the ground truth must be established and validated. The novel design of Tangerine also considers this, and is the first framework which provides an effective ground truth development technique through correlation with third party CTI services. In addition, this thesis divulges the design detail of the Clementine module. This module is responsible for the online detection of emerging threats through the utilisation of machine learning algorithms and integration with cluster computation technology. Critically, this module enables the detection of previously unobserved attacks emanating from nodes within a high-speed network.

Developing upon the design of this system, this thesis outlines the implementation detail of a proof-of-concept intrusion detection framework. This implementation is created with the intention to satisfy the requirements outlined in Section 3.1. Furthermore, the implementation is released publicly[1] in an open-source manner to promote further examination and research into the cyber security field. A wide range of industry beneficiaries could also adopt this framework for network defense purposes in large scale on-premises or cloud infrastructures. In detail, Citrus provides companies the ability to create a bespoke labelled data set, which contains a normal traffic profile relating to actual behaviour encountered within their network. Using this tailored data set and Citrus, companies could also defend their network against emerging threats through Citrus' inherent scalable anomaly detection capabilities.

An evaluation is also performed within this thesis which attempts to assess whether the aforementioned design requirements have been met through experimentation and analysis. In detail, the evaluation explores Citrus' ability to successfully develop a ground truth and perform near real-time intrusion detection through the utilisation of well-established machine learning algorithms. For this purpose, an appropriate network test-bed located within Lancaster University was chosen. Citrus, alongside essential network services, are deployed within this environment to evaluate its ability using real traffic captured from within.

In addition to the contributions related to the public release of Citrus, this thesis has also contributed to the public release of a novel intrusion detection data set. As previously outlined, the Tangerine component within Citrus orchestrates the collection and labelling of flows captured from various honeypot deployments. Through this process, telemetry representative of emerging attack vectors is compiled into a data set appropriate for the evaluation of next-generation anomaly detection approaches to intrusion detection. This data set is publicly released[2] under open-source license in an open and reusable format. Similarly to the open-source nature of Citrus, the data set is delivered in such a manner to be adopted by third parties within both academia and industry for the evaluation of next-

[1]https://github.com/ruzzzzz/Citrus
[2]https://github.com/ruzzzzz/LUFlow

generation attack detection techniques.

As previously discussed, Citrus has achieved Design Requirements 1 and 2 through the analysis, public release, and updating of the LUFlow '20 data set. As a result, we have also demonstrated that honeypots can be used as means to capture emerging attack patterns and author a continuously updated intrusion detection data set, thus, positively answering Research Question 1.

Through a series of evaluations, a number of beneficial properties relating to Citrus are identified. Initially, an investigation into the consistency of the clusters identified through the calculation of graph-based features is conducted to ensure an accurate ground truth is developed. The silhouette metric is chosen for this purpose, and a time-series of these values are presented for every day in which telemetry is captured and labelled using the ground truth developed for that specific day. This evaluation highlighted clear consistency in all examined clusters through wholly positive silhouette values, indicating that nodes labelled as malicious do not actually belong to clusters associated with outlier telemetry traces. As a result of this validation of the ground truth, Design Requirement 3 has been successfully achieved. Due to this, we are able to provide an answer for Research Question 2. Specifically, this thesis has demonstrated that CTI services can be successfully leveraged to develop and validate a ground truth for telemetry captured by honeypots.

In addition, an examination of Citrus' ability to efficiently and effectively detect attacks through both offline and online detection approaches is conducted. This range of evaluations has demonstrated the accuracy of Citrus' intrusion detection mechanism in a real-world practical setting through the successful classification of previously unobserved attacks orchestrated against victim servers. As well as demonstrating the accuracy of the approach, the design of Citrus has also been shown to be highly scalable, capable of the classification of thousands of live flows emanating from real networked devices within seconds, thus, achieving near real-time prediction. As previously discussed, the results extracted from these evaluations have shown that Design Requirement 4 of Citrus has been met. Since we have demonstrated that this design requirement is met, we can also positively answer Research Question 3. In detail, this thesis has established that Big Data technologies can be leveraged to perform accurate intrusion detection through the prediction of vast amounts of traffic in near real-time.

6.1.1 Summary

To summarise, this thesis has made the following contributions:

- The design of a framework which provides the practical integration of CTI for active network defense. This design enables the establishment and validation of a robust ground truth to aid towards accurate anomaly detection.
- The design of a near real-time anomaly detection approach, which exploits properties of data and system parallelism.
- The realisation of both of these designs to create an open-source prototype implementation of an innovative intrusion detection framework.
- A comprehensive evaluation of this prototype. This was made possible through the use of a realistic network environment.
- The public release of a continuously updated novel intrusion detection data set which is supported by a robust ground truth.

6.2 Limitations

The work presented in thesis has a number of limitations. These relate to the approaches taken in the design of system architecture, application implementation, and classification methods.

One such limitation is the method used to identify malicious activity within honeypot telemetry data. As mentioned previously, CTI services are used for this process. Due to the nature of CTI services, they are likely to capture overtly intrusive and noisey attacks. As a result, if stealthy attacks were orchestrated on the honeypots, it is possible that they have not been observed elsewhere by CTI services and the corresponding telemetry would *not* be labelled as malicious. The decision making process of IDSs which use this telemetry as training data will be impacted accordingly.

In a similar vein to the previous limitation, the ML algorithms adopted by Citrus could also be tricked via adversarial ML attacks. Adversarial ML attacks are manifested in multiple ways, but a common method is data poisoning. Since the labels of the training data are dictated by CTI services, Citrus trusts that the

CTI services are truthful. Many of the CTI services utilised by Citrus leverage user submissions and are likely not verified. As a result, there is a possibility that the honeypot telemetry data could be poisoned to provide desired results by attackers.

Another important limitation is the different types of honeypots used. Due to restrictions within the experimental environment, only low and medium-interaction honeypots were deployed. In contrast to high-interaction honeypots, these types of honeypots do not allow observation of the entire attack life cycle. In the case that high-interaction honeypots were leveraged, there is a strong possibility that more realistic data would have been captured and that the data would also better reflect attack patterns of a more sophisticated nature.

6.3 Future Work

This thesis has outlined the design, implementation, and evaluation of a next-generation network security framework, Citrus, which is adept at near real-time detection of emerging threats. Despite the promising results discussed in the evaluation, there are a number of additional avenues, which could be explored to make further contributions to the cyber security field.

One such area proposed for future work is the automatic revision and re-training of machine learning models to deal with behavioural changes apparent within captured telemetry. As previously discussed, the modern threat landscape consists of myriad threat actors consistently producing an evolving array of innovative attack techniques. As the data set composed by Citrus is constantly receiving updates due to the automatic nature of Citrus' collection and labelling mechanism, it is able to reflect these rapidly evolving behavioural traffic patterns. The incremental update of machine learning models has been shown to ensure that defense mechanisms are able to detect the most novel and innovative threats [94].

Another way in which this work can be extended is the creation of automated remediation measures. While Citrus has been demonstrably shown to be able to detect emerging threats, malicious classifications by Citrus are intended to alert network administrators of attacks. These administrators will then need to apply remedial actions manually in order to neutralise the threat. A system which automatically performs remediation, such as the blocking of network flows from a particular source address, will ensure attacks are thwarted before they can perform further malicious actions. An avenue in which this could be addressed is Software-defined Networking (SDN), which enables the programmatic manipulation of network behaviour without the need for specialised hardware.

The LUFlow '20 data set incorporates a variety of normal and malicious network flows. As network flows are captured using honeypots, all communication with them should be treated as suspicious since it is unsolicited. Many of these flows are labelled as malicious as they are identified by CTI services as performing malicious actions elsewhere. However, there exist some flows that are not labelled as malicious and are instead labelled as outliers. Outlier flows are interesting as they have initiated communication with honeypots, yet are not known to CTI services. There could be many possible explanations for this, including that they could be stealthy, targeted attacks. As a result, there could be value in the research of such flows to identify their true nature. Such investigation could reveal previously unobserved infiltration techniques.

References

[1] Cyber Security Ventures Cybercrime Report. URL: `https://cybersecurityventures.com/2015-wp/wp-content/uploads/2017/10/2017-Cybercrime-Report.pdf/`.

[2] CISCO. Cisco Annual Internet Report (2018–2023) White Paper. 2020.

[3] Mohammad Mehedi Hassan, Abdu Gumaei, Ahmed Alsanad, Majed Alrubaian, and Giancarlo Fortino. A hybrid deep learning model for efficient intrusion detection in big data environment. *Information Sciences*, 513:386–396, March 2020.

[4] Muhammet Baykara and Resul Das. A novel honeypot based security approach for real-time intrusion detection and prevention systems. *Journal of Information Security and Applications*, 41:103–116, August 2018.

[5] Riyaz Ahamed Ariyaluran Habeeb, Fariza Nasaruddin, Abdullah Gani, Mohamed Ahzam Amanullah, Ibrahim Abaker Targio Hashem, Ejaz Ahmed, and Muhammad Imran. Clustering-based real-time anomaly detection—A breakthrough in big data technologies. *Transactions on Emerging Telecommunications Technologies*, pages 1–27, 2019.

[6] Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. *META Group*, February 2001.

[7] Richard Zuech, Taghi M Khoshgoftaar, and Randall Wald. Intrusion detection and Big Heterogeneous Data: a Survey. *Journal of Big Data*, 2:3, 2015.

[8] Elisabetta Raguseo. Big data technologies: An empirical investigation on their adoption, benefits and risks for companies. *International Journal of Information Management*, 38:187–195, 2018.

[9] Peter Clay. A modern threat response framework. *Network Security*, 2015, 04 2015.

[10] Apache Hadoop. URL: `https://hadoop.apache.org/`.

[11] BigTable. URL: `https://cloud.google.com/bigtable/`.

[12] Elasticsearch. URL: `https://www.elastic.co/`.

[13] Apache Spark. URL: `https://spark.apache.org/`.

[14] Valerio Morfino and Salvatore Rampone. Towards near-real-time intrusion detection for IoT devices using supervised learning and apache spark. *Electronics (Switzerland)*, 9(3), 2020.

[15] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Proceedings of NSDI 2012: 9th USENIX Symposium on Networked Systems Design and Implementation*, pages 15–28, 2012.

[16] Muhammet Baykara and Resul Daş. A Survey on Potential Applications of Honeypot Technology in Intrusion Detection Systems. *International Journal of Computer Networks and Applications (IJCNA)*, 2(5).

[17] Shodan. URL: `https://shodan.io/`.

[18] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation. *Proceedings of the 1st Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2011*, pages 29–36, 2011.

[19] KDD '99 Data Set. URL: `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html/`.

[20] Hanan Hindy, David Brosset, Ethan Bayne, Amar Seeam, Christos Tachtatzis, Robert Atkinson, and Xavier Bellekens. A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets. 1(1), 2018.

[21] Unified Host and Network Dataset. URL: `https://csr.lanl.gov/data/2017.html/`.

[22] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[23] Sebastian Abt and Harald Baier. Are We Missing Labels? A Study of the Availability of Ground-Truth in Network Security Research. *Proceedings - 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2014*, pages 40–55, 2016.

[24] Y. Chen, X. Lian, D. Yu, S. Lv, S. Hao, and Y. Ma. Exploring shodan from the perspective of industrial control systems. *IEEE Access*, 8:75359–75369, 2020.

[25] GreyNoise Intelligence. URL: `https://greynoise.io/`.

[26] Censys. URL: `https://censys.io/`.

[27] FireEye Threat Trend Report 2020. URL: `https://content.fireeye.com/m-trends/rpt-m-trends-2020`.

[28] Mitre ATT&CK. URL: `https://attack.mitre.org/matrices/enterprise/`.

[29] R. Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security Privacy*, 9(3):49–51, 2011.

[30] Savita Mohurle and Manisha Patil. A brief study of wannacry threat: Ransomware attack 2017. *International Journal of Advanced Research in Computer Science*, 8(5), 2017.

[31] Rob Joyce. Disrupting Nation State Hackers. *USENIX Association*, January 2016.

[32] Subash Poudyal and Dipankar Dasgupta. Ai-powered ransomware detection framework. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1154–1161, 2020.

[33] CVE-2018-15982. URL: `https://nvd.nist.gov/vuln/detail/CVE-2018-15982/`.

[34] Michael Hopkins and Ali Dehghantanha. Exploit kits: The production line of the cybercrime economy? In *2015 second international conference on Information Security and Cyber Forensics (InfoSec)*, pages 23–27. IEEE, 2015.

[35] CVE-2018-8174. URL: `https://nvd.nist.gov/vuln/detail/CVE-2018-8174/`.

[36] Malwarebytes 2019 Exploit Kit Review. URL: `https://blog.malwarebytes.com/exploits-and-vulnerabilities/2019/11/exploit-kits-fall-2019-review/`.

[37] Muhammad N Sakib and Chin-Tser Huang. Using anomaly detection based techniques to detect http-based botnet c&c traffic. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[38] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647, 2009.

[39] Sudipta Chowdhury, Mojtaba Khanzadeh, Ravi Akula, Fangyan Zhang, Song Zhang, Hugh Medal, Mohammad Marufuzzaman, and Linkan Bian. Botnet detection using graph-based feature clustering. *Journal of Big Data*.

[40] Mariposa White Paper. URL: `https://defintel.com/docs/Mariposa_White_Paper.pdf`.

[41] Angelos K. Marnerides and Andreas U. Mauthe. Analysis and characterisation of botnet scan traffic. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pages 1–7, 2016.

[42] Nmap. URL:
https://nmap.org/book/man-port-scanning-techniques.html.

[43] ENISA Cryptojacking Threat Landscape 2020. URL:
https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-cryptojacking.

[44] Ronny Richardson and Max M North. Ransomware: Evolution, Mitigation and Prevention. *International Management Review*, 13(1):10, 2017.

[45] Savita Mohurle and Manisha Patil. A brief study of Wannacry Threat: Ransomware Attack 2017. *International Journal of Advanced Research in Computer Science*, 8(5).

[46] Guy Martin, Saira Ghafur, James Kinross, Chris Hankin, and Ara Darzi. WannaCry - A year on. *BMJ (Online)*, 361(June):10–11, 2018.

[47] DoublePulsar Analysis. URL: https://zerosum0x0.blogspot.com/2017/04/doublepulsar-initial-smb-backdoor-ring.html#pulsar_step0.

[48] Hackers are using NSA's DoublePulsar backdoor in attacks. URL:
https://www.securityweek.com/hackers-are-using-nsas-doublepulsar-backdoor-attacks.

[49] MS17-010 Security Bulletin. URL: https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010.

[50] Cisco Teslacrypt. URL:
https://blogs.cisco.com/security/talos/teslacrypt.

[51] Aurelien Palisse, Helene Le Bouder, and Jean-Louis Lanet. Ransomware and the Legacy Crypto API. volume 10158 of *CRiSIS 2016*, pages 11–28. Springer International Publishing, 2017.

[52] Eric Hutchins, Michael Cloppert, and Rohan Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare Security Research*, 1, 01 2011.

[53] Gobuster. URL: https://github.com/OJ/gobuster.

[54] Dennis Mirante and Justin Cappos. Understanding password database compromises. *Dept. of Computer Science and Engineering Polytechnic Inst. of NYU, Tech. Rep. TR-CSE-2013-02*, 2013.

[55] Ping Chen, Lieven Desmet, and Christophe Huygens. A Study on Advanced Persistent Threats. pages 63–72, 2014.

[56] GTFOBins. URL: `https://gtfobins.github.io/`.

[57] CSO Cyber Kill Chain Report. URL: https://www.csoonline.com/article/2134037/strategic-planning-erm-the-practicality-of-the-cyber-kill-chain-approach-to-security.html/.

[58] Saurabh Singh, Pradip Kumar Sharma, Seo Yeon Moon, Daesung Moon, and Jong Hyuk Park. A comprehensive study on APT attacks and countermeasures for future networks and communications: challenges and solutions. *The Journal of Supercomputing*, pages 1–32, sep 2016.

[59] Mitko Bogdanoski, Tomislav Shuminoski, and Aleksandar Risteski. Analysis of the syn flood dos attack. *International Journal of Computer Network and Information Security*, 5:1–11, 06 2013.

[60] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE Communications Surveys Tutorials*, 15(4):2046–2069, 2013.

[61] Netscout Threat Intelligence Report 2020. URL: `https://www.netscout.com/threatreport`.

[62] Akamai State of the Internet / Security Report. URL: `https://www.akamai.com/uk/en/multimedia/documents/state-of-the-internet/soti-security-a-year-in-review-report-2020.pdf`.

[63] Abdallah Ghourabi, Tarek Abbes, and Adel Bouhoula. Characterization of attacks collected from the deployment of Web service honeypot. *Security and Communication Networks*, 7(2):338–351, feb 2014.

[64] Ibrahim Yahya, Mohammed Al, Prashant Chauhan, Shivi Shukla, and M B Potdar. Review on efficient log analysis to evaluate multiple honeypots using ELK. (6):492–504, 2016.

[65] Glutton. URL: `https://github.com/mushorg/glutton`.

[66] Heralding. URL: `https://github.com/johnnykv/heralding`.

[67] Cowrie. URL: `https://github.com/cowrie/cowrie/`.

[68] Dionaea. URL: `https://github.com/DinoTools/dionaea/`.

[69] libemu. URL: `https://github.com/buffer/libemu/`.

[70] Martin Valicek, Gregor Schramm, Martin Pirker, and Sebastian Schrittwieser. Creation and Integration of Remote High Interaction Honeypots. *Proceedings - 2017 International Conference on Software Security and Assurance, ICSSA 2017*, pages 50–55, 2018.

[71] Stewart Sentanoe, Benjamin Taubmann, and Hans P. Reiser. Virtual machine introspection based ssh honeypot. In *Proceedings of the 4th Workshop on Security in Highly Connected IT Systems*, SHCIS '17, page 13–18, New York, NY, USA, 2017. Association for Computing Machinery.

[72] Libpcap. URL: `https://www.tcpdump.org/manpages/pcap.3pcap.html/`.

[73] Monowar H Bhuyan, Dhruba K Bhattacharyya, and Jugal K Kalita. Towards Generating Real-life Datasets for Network Intrusion Detection. 17(6):683–701, 2015.

[74] Martin Husák and Martin Drašar. Flow-based Monitoring of Honeypots. pages 63–70, 2013.

[75] Dominic Storey. Catching flies with honey tokens. *Network Security*, 2009(11):15–18, 2009.

[76] Ryan Mills, Nicholas Race, and Matthew Broadbent. Citrus: Orchestrating security mechanisms via adversarial deception. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–4, 2020.

[77] Emmanouil Vasilomanolakis, Shreyas Srinivasa, Carlos Garcia Cordero, and Max Mühlhäuser. Multi-stage attack detection and signature generation with ICS honeypots. In *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP*

Network Operations and Management Symposium, pages 1227–1232. Institute of Electrical and Electronics Engineers Inc., June 2016.

[78] Bro. URL: `https://www.zeek.org/`.

[79] Daisuke Miyamoto, Satoru Teramura, and Masaya Nakayama. Intercept: High-interaction server-type honeypot based on live migration. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '14, page 147–152, Brussels, BEL, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[80] Iik Muhamad Malik Matin and Budi Rahardjo. Malware Detection Using Honeypot and Machine Learning. *2019 7th International Conference on Cyber and IT Service Management, CITSM 2019*, 2019.

[81] Hyrum S. Anderson and Phil Roth. EMBER: an open dataset for training static PE malware machine learning models. *CoRR*, abs/1804.04637, 2018.

[82] E Alata, M Dacier, Y Deswarte, M Kaaâniche, K Kortchinsky, V Nicomette, V H Pham, and F Pouget. Collection and analysis of attack data based on honeypots deployed on the Internet. In Dieter Gollmann, Fabio Massacci, and Artsiom Yautsiukhin, editors, *Quality of Protection*, pages 79–91, Boston, MA, 2006. Springer US.

[83] Florian Skopik, Giuseppe Settanni, and Roman Fiedler. A problem shared is a problem halved: A survey on the dimensions of collective cyber defense through security information sharing. *Computers and Security*, 60:154–176, jul 2016.

[84] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):20, 2019.

[85] G. Creech and J. Hu. A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. *IEEE Transactions on Computers*, 63(4):807–819, 2014.

[86] Tarrah R Glassvandetr, ornlgov Iannacone, Michael D Iannaconemd, ornlgov Vincent, Maria S Vincentms, ornlgov Chen, and Robert A Bridgesra. A Survey of Intrusion Detection Systems Leveraging Host Data. 2018.

[87] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An overview of ip flow-based intrusion detection. *IEEE Communications Surveys Tutorials*, 12(3):343–356, 2010.

[88] Symantec Internet Security Threat Report 2017. URL: `https://docs.broadcom.com/doc/istr-22-2017-en`.

[89] A. Alazab, M. Hobbs, J. Abawajy, and M. Alazab. Using feature selection for intrusion detection system. In *2012 International Symposium on Communications and Information Technologies (ISCIT)*, pages 296–301, 2012.

[90] Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck. Learning Intrusion Detection: Supervised or Unsupervised? In Fabio Roli and Sergio Vitulano, editors, *Image Analysis and Processing – ICIAP 2005*, pages 50–57, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[91] Phurivit Sangkatsanee, Naruemon Wattanapongsakorn, and Chalermpol Charnsripinyo. Practical real-time intrusion detection using machine learning approaches. *Computer Communications*, 34(18):2227–2235, 2011.

[92] Vigneswaran K. Rahul, R. Vinayakumar, Kp Soman, and Prabaharan Poornachandran. Evaluating Shallow and Deep Neural Networks for Network Intrusion Detection Systems in Cyber Security. *2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2018*, pages 1–6, 2018.

[93] Antonio G P Lobato, Martin Andreoni Lopez, and Alvaro A Cardenas. A Fast and Accurate Threat Detection and Prevention Architecture using Stream Processing. pages 1–12.

[94] Eduardo Viegas, Altair Santin, Alysson Bessani, and Nuno Neves. BigFlow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. *Future Generation Computer Systems*, 93:473–485, 2019.

[95] Apache Flink. URL: `https://flink.apache.org/`.

[96] Mawi archive. URL: `http://mawi.wide.ad.jp/mawi/samplepoint-F/`.

[97] Juliette Dromard and Philippe Owezarski. Study and Evaluation of Unsupervised Algorithms Used in Network Anomaly Detection. *Advances in Intelligent Systems and Computing*, 1070:397–416, 2020.

[98] Apache Kafka. URL: `https://kafka.apache.org/`.

[99] M. Mazhar Rathore, Anand Paul, Awais Ahmad, Seungmin Rho, Muhammad Imran, and Mohsen Guizani. Hadoop based real-time intrusion detection for high-speed networks. *2016 IEEE Global Communications Conference, GLOBECOM 2016 - Proceedings*, (Ml):0–5, 2016.

[100] Iman Sharafaldin, Amirhossein Gharib, Arash Habibi Lashkari, and Ali Ghorbani. Towards a reliable intrusion detection benchmark dataset. *Software Networking*, 2017:177–200, 01 2017.

[101] Nour Moustafa and Jill Slay. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., dec 2015.

[102] Nasrin Sultana, Naveen Chilamkurti, Wei Peng, and Rabei Alhadad. Survey on SDN based network intrusion detection system using machine learning approaches. *Peer-to-Peer Networking and Applications*, 12(2):493–501, 2019.

[103] Jérôme François, Shaonan Wang, Radu State, and Thomas Engel. BotTrack: Tracking botnets using netflow and pageRank. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6640 LNCS(PART 1):1–14, 2011.

[104] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, 2018.

[105] Przemysław Bereziński, Bartosz Jasiul, and Marcin Szpyrka. An Entropy-Based Network Anomaly Detection Method. 17(4):2367–2408, April 2015.

[106] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Flow-based benchmark data sets for intrusion detection. In *Proceedings of the 16th European conference on cyber warfare and security*, pages 361–369, 2017.

[107] Marek Małowidzki, Przemysław Berezi, and Michał Mazur. Network Intrusion Detection: Half a Kingdom for a Good Dataset. *Proceedings of NATO STO SAS-139 Workshop*, pages 1–6, 2015.

[108] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. Flow-based intrusion detection: Techniques and challenges. *Computers & Security*, 70:238–254, 2017.

[109] Alessandro D'Alconzo, Idilio Drago, Andrea Morichetta, Marco Mellia, and Pedro Casas. A Survey on Big Data for Network Traffic Monitoring and Analysis. *arXiv*, 16(3):800–813, 2020.

[110] Prasanta Gogoi, Monowar H Bhuyan, D K Bhattacharyya, and J K Kalita. Packet and Flow Based Network Intrusion Dataset. In Manish Parashar, Dinesh Kaushik, Omer F Rana, Ravi Samtaney, Yuanyuan Yang, and Albert Zomaya, editors, *Contemporary Computing*, pages 322–334, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[111] Nam Nguyen and Rich Caruana. Classification with partial labels. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 551–559, 2008.

[112] W. Meng, Y. Liu, S. Zhang, D. Pei, H. Dong, L. Song, and X. Luo. Device-agnostic log anomaly classification with partial labels. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–6, 2018.

[113] Anna Sperotto, Ramin Sadre, Frank Van Vliet, and Aiko Pras. A labeled data set for flow-based intrusion detection. *Lecture Notes in Computer*

Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5843 LNCS:39–50, 2009.

[114] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. MAWILab : Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking. 2010.

[115] Francisco J. Aparicio-Navarro, Konstantinos G. Kyriakopoulos, and David J. Parish. Automatic dataset labelling and feature selection for intrusion detection systems. In *Proceedings - IEEE Military Communications Conference MILCOM*, pages 46–51. Institute of Electrical and Electronics Engineers Inc., November 2014.

[116] Airpwn-ng. URL: `https://github.com/ICSec/airpwn-ng/`.

[117] Francesco Gargiulo, Claudio Mazzariello, and Carlo Sansone. Automatically building datasets of labeled IP traffic traces: A self-training approach. *Applied Soft Computing Journal*, 12(6):1640–1649, 2012.

[118] John Mchugh. Testing Intrusion Detection Systems : A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. 3(4):262–294, 2001.

[119] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A Detailed Analysis of the KDD CUP 99 Data Set. (Cisda):1–6, 2009.

[120] Kristopher Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Defense Technical Information Center, 1999.

[121] Matthew V Mahoney and Philip K Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In Giovanni Vigna, Christopher Kruegel, and Erland Jonsson, editors, *Recent Advances in Intrusion Detection*, pages 220–237, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[122] Tcpdump. URL: `https://www.tcpdump.org/`.

[123] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security*, 31(3):357–374, 2012.

[124] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Computers and Security*, 45:100–123, 2014.

[125] CICFlowMeter. URL:
`https://github.com/ahlashkari/CICFlowMeter/`.

[126] Ranjit Panigrahi and Samarjeet Borah. A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *International Journal of Engineering and Technology(UAE)*, 7(3.24 Special Issue 24):479–482, 2018.

[127] Joffrey L Leevy, Taghi M Khoshgoftaar, Richard A Bauder, and Naeem Seliya. A survey on addressing high-class imbalance in big data. *Journal of Big Data*, 5(1):42, 2018.

[128] Rushi Longadge and Snehalata Dongre. Class imbalance problem in data mining review. *International Journal of Computer Science and Network*, 2, 2013.

[129] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers and Security*, 86:147–167, 2019.

[130] Faranak Abri, Sima Siami-Namini, Mahdi Adl Khanghah, Fahimeh Mirza Soltani, and Akbar Siami Namin. The performance of machine and deep learning classifiers in detecting zero-day vulnerabilities. *CoRR*, abs/1911.09586, 2019.

[131] Chieh Yen Lin, Cheng Hao Tsai, Ching Pei Lee, and Chih Jen Lin. Large-scale logistic regression and linear support vector machines using spark. *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014*, pages 519–528, 2015.

[132] Mustapha Belouch, Salah El Hadaj, and Mohamed Idlianmiad. Performance evaluation of intrusion detection based on machine learning using apache spark. *Procedia Computer Science*, 127:1–6, 2018.

[133] Manish Kulariya, Priyanka Saraf, Raushan Ranjan, and Govind P. Gupta. Performance analysis of network intrusion detection schemes using Apache Spark. *International Conference on Communication and Signal Processing, ICCSP 2016*, pages 1973–1977, 2016.

[134] Hao Zhang, Shumin Dai, Yongdan Li, and Wenjun Zhang. Real-time Distributed-Random-Forest-Based Network Intrusion Detection System Using Apache Spark. In *2018 IEEE 37th International Performance Computing and Communications Conference, IPCCC 2018*. Institute of Electrical and Electronics Engineers Inc., jul 2018.

[135] Qianru Zhou and Dimitrios Pezaros. Evaluation of Machine Learning Classifiers for Zero-Day Intrusion Detection-An Analysis on CIC-AWS-2018 dataset. *CoRR*, abs/1905.03685, 2019.

[136] Gokul Kannan Sadasivam, Chittaranjan Hota, and Bhojan Anand. Detection of Severe SSH Attacks Using Honeypot Servers and Machine Learning Techniques. *Software Networking*, 2017(1):79–100, 2017.

[137] Sana Siddiqui, Muhammad Salman Khan, Ken Ferens, and Witold Kinsner. Detecting advanced persistent threats using fractal dimension based machine learning classification. In *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, IWSPA '16, page 64–69, New York, NY, USA, 2016. Association for Computing Machinery.

[138] Marzia Zaman and Chung Horng Lung. Evaluation of machine learning techniques for network intrusion detection. *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, pages 1–5, 2018.

[139] GitHub. URL: `https://github.io/`.

[140] Python. URL: `https://python.org/`.

[141] Apache Mesos. URL: `http://mesos.apache.org/`.

[142] Kubernetes. URL: `https://kubernetes.io/`.

[143] PySpark. URL: `https://pypi.org/project/pyspark/`.

[144] Spark Streaming. URL: `http://spark.apache.org/docs/latest/streaming-programming-guide.html/`.

[145] Amazon Web Services Kinesis. URL: `https://aws.amazon.com/kinesis/`.

[146] Spark RDD Guide. URL: `http://spark.apache.org/docs/2.4.4/rdd-programming-guide.html`.

[147] Matei Zaharia. Discretized Streams. *An Architecture for Fast and General Data Processing on Large Clusters*, (1):65, 2016.

[148] grequests. URL: `https://pypi.org/project/grequests/`.

[149] Requests Library. URL: `https://pypi.org/project/requests/`.

[150] gevent. URL: `http://www.gevent.org/`.

[151] Hybrid Analysis. URL: `https://www.hybrid-analysis.com/`.

[152] NetworkX. URL: `https://networkx.github.io/`.

[153] scikit-learn. URL: `https://scikit-learn.org/`.

[154] LUFlow '20 Intrusion Detection Data Set. URL: `https://github.com/ruzzzzz/LUFlow`.

[155] HdfsCLI. URL: `https://pypi.org/project/hdfs/`.

[156] Angelos Marnerides, Daniel Prince, John Couzins, Ryan Mills, Vasileios Giotsas, Paul McEvatt, David Markham, and Catherine Irvine. Fujitsu white paper: Cyber threat lab, 2019.

[157] VMware. URL: `https://www.vmware.com/`.

[158] TPot Honeypot. URL: `https://github.com/dtag-dev-sec/tpotce`.

[159] Joy. URL: `https://github.com/cisco/joy/`.

[160] Logstash. URL: `https://www.elastic.co/logstash`.

[161] Geolite2 database. URL:
`https://dev.maxmind.com/geoip/geoip2/geolite2-asn-csv-database/`.

[162] Elasticpot Honeypot. URL:
`https://gitlab.com/bontchev/elasticpot/`.

[163] CVE-2020-14404. URL:
`https://ubuntu.com/security/notices/USN-4573-1`.

[164] CVE-2020-1350. URL: `https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2020-1350`.

[165] Cve-2020-1350 Proof of Concept. URL:
`https://github.com/Plazmaz/CVE-2020-1350-poc`.

[166] Peter J Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[167] Tuan-Hong Chua and Iftekhar Salam. Evaluation of machine learning algorithms in network-based intrusion detection system. Cryptology ePrint Archive, Paper 2022/335, 2022. `https://eprint.iacr.org/2022/335`.

[168] Sophos Threat Report 2020. URL:
`https://www.sophos.com/en-us/labs/security-threat-report.aspx`.

[169] rockyou.txt Wordlist. URL: `https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt`.

[170] K. Saikawa and V. Klyuev. Detection and classification of malicious access using a dionaea honeypot. In *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 2, pages 844–848, 2019.

[171] Sadegh Torabi, Elias Bou-Harb, Chadi Assi, El Mouatez Billah Karbab, Amine Boukhtouta, and Mourad Debbabi. Inferring and Investigating IoT-Generated Scanning Campaigns Targeting A Large Network Telescope. *IEEE Transactions on Dependable and Secure Computing*, pages 1–17, 2020.

[172] Kali Linux. URL: `https://www.kali.org/`.

[173] Meterpreter. URL: `https://www.offensive-security.com/metasploit-unleashed/about-meterpreter`.

[174] XMRig. URL: `https://github.com/xmrig/xmrig`.

[175] Dionaea DoublePulsar emulation. URL: `https://github.com/DinoTools/dionaea/issues/240`.

[176] SecurityIntelligence WannaCry Report. URL: https://securityintelligence.com/news/weekly-security-news-roundup-wannacry-dominated-ransomware-detections-in-q1-2020/.

[177] VirusTotal. URL: `https://virustotal.com/`.