MINOS: UNSUPERVISED NETFLOW-BASED DETECTION OF INFECTED AND ATTACKED HOSTS, AND ATTACK TIME IN LARGE NETWORKS

by

Mousume Bhowmick

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Computer Science

Boise State University

August 2019

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Mousume Bhowmick

Thesis Title: MINOS: Unsupervised Netflow-based Detection of Infected and Attacked Hosts, and Attack Time in Large Networks

Date of Final Oral Examination: 30 April 2019

The following individuals read and discussed the thesis submitted by student Mousume Bhowmick, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Dianxiang Xu, Ph.D.	Chair, Supervisory Committee
Edoardo Serra, Ph.D.	Co-Chair, Supervisory Committee
Francesca Spezzano, Ph.D.	Member, Supervisory Committee

The final reading approval of the thesis was granted by Dianxiang Xu, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.

DEDICATON

Dedicated to all women who died a lot.

ACKNOWLEDGEMENTS

I am wholeheartedly thankful to Boise State University for allowing me to pursue graduate study. I would also like to express the highest gratitude to the Office of Information Technology of Boise State University for providing a real dataset and valuable information for this research.

It is my great fortune to get a chance as a research assistant in Dr. Dianxiang Xu's team: Software Engineering and Assurance Lab (SEAL). I highly respect the learning opportunity and environment he provided me. I am grateful and have great appreciation to him because of all supports and keeping faith with me. Taking software engineering and software security courses under his supervision built a strong foundation towards my career path. Moreover, his achievements, dedication, and passion for the research domain in Computer Security will ever be a constant source of motivation for my entire life.

I would also like to thank Dr. Edoardo Serra, who guided me since the beginning of this thesis. I would not hesitate to confess that I had no knowledge about machine learning or data science, which I learned in-depth from his courses. His outstanding teaching style makes things more convenient and encourages me to research a problem that related to the data science field. He has helped me to identify my weaknesses and push my abilities beyond the boundaries through continuous leadership and uncountable productive meetings in the last two years. Last but not least, I am expressing humble gratitude to him because of allowing me to knock on his door at any time.

I am also grateful to Dr. Francesca Spezzano for being in my thesis committee and providing valuable feedback for my work. Her insightful comments on my research work helped me to develop better ideas.

Finally, I am grateful to my family for always supporting and helping me to realize the importance of education. Special thanks to my beloved husband for being amazing emotional support including the courage to be patient at the difficult times. I would not have come up to this level without their neverending encouragement and love.

ABSTRACT

Monitoring large-scale networks for malicious activities is increasingly challenging: the amount and heterogeneity of traffic hinder the manual definition of IDS signatures and deep packet inspection. In this thesis, we propose MINOS, a novel fully unsupervised approach that generates an anomaly score for each host allowing us to classify with high accuracy each host as either infected (generating malicious activities), attacked (under attack), or clean (without any infection). The generated score of each hour is able to detect the time frame of being attacked for an infected or attacked host without any prior knowledge. MINOS automatically creates a personalized traffic behavioral model for each host and does not require any previous knowledge of existing or unknown attacks. Experimental evaluation on a real large academic network over one year of data shows that MINOS achieves very high accuracy, even when analyzing only two weeks of data. We demonstrate MINOS is also efficient and faster than a state-of-the-art approach for unsupervised anomaly detection on traffic data.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IDS	Intrusion Detection Systems
HIDS	Host (based) Intrusion Detection Systems
NIDS	Network (based) Intrusion Detection Systems
SIDS	Signature (based) Intrusion Detection Systems
AIDS	Anomaly (based) Intrusion Detection Systems
MLT	Machine Learning Technique
GMM	Gaussian Mixture Model
BGMM	Bayesian Gaussian Mixture Model
OC-SVM	One Class Support Vector Machine
SVM	Support Vector Machine
EM	Expectation Maximization
LSTM	Long Short Term Memory
AUROC	Area Under Receiver Operating Characteristic
AP	Average Precision
CH	Clean Host
HUA	Host Under Attack
MH	Malicious Host
UDP	User Datagram Protocol
ESP	Encapsulating Security Payload
TCP	Transmission Control Protocol
GRE	Generic Routing Encapsulation
IPv6	Internet Protocol Version (6)

SCTP	Stream Control Transmission Protocol
ICMP	Internet Control Message Protocol
IPFIX	Internet Protocol Flow Information Export
DPI	Deep Packet Inspection

CHAPTER ONE: INTRODUCTION

1.1 Motivation

The Internet is a widespread system in continuous evolution, where the number of attacks, Internet traffic, and line speed continues to grow [32]. Nowadays, it is common to use an access speed of 1 - 10 Gbps. Since bandwidth for wired connections is available, high-bandwidth services are being offered to users. For example, a university network reaches traffic averages in the order of hundreds of Mbps, including high activity peaks in the order of Gbps [53, 58, 59]. On backbone networks, the throughput will even be higher. Also, it is conventional for Internet users to have been a victim of an attack because of attackers' constant assaults into networked systems. For example, a hacked machine can send out sensitive data to an unauthorized host; in this case, the cost of these attacks would be billions of U.S. dollars [15]. Therefore, it becomes significant to detect and prevent these intrusions as early as possible. Therefore, Network Intrusion Detection Systems (NIDS) need to handle the rising number of attacks, the growth of Internet traffic as well as the increase in line speed.

The most popular systems such as Bro [46], SNORT[1], and Suricata[2] demonstrate high resource consumption, when confronted with the vast amount of data found in today’s high-speed networks [14]. Additionally, those systems are doing an in-depth analysis of the packets. If the packet’s data is encrypted, then it poses a new challenge to payload-based systems. Moreover, researchers assess that the payload-based NIDS processing capability lies between 100 Mbps and 200 Mbps [17, 34], which is inconvenient to this era. In contrast, the flowbased NIDS looks at aggregated information of related packets in the form of flow, so the amount of analyzing data is reduced [1, 51, 53]. In this context, flow-based approaches might be a promising candidate for Intrusion Detection research [59].

Traffic networks of large organizations are challenging to protect and monitor, due to the increasing amount of communications and heterogeneity of user behaviors and devices. Misuse-based systems (e.g., IDS [46]) require a priori knowledge on attacks and standard definitions of signatures by security analysts. Therefore, researchers focused on building statistical anomaly-based systems [9]. In this context, proposed supervised models often train on traffic datasets that contain artifacts (e.g., DARPA datasets [37]). As a result, those models do not generalize well when deployed in the real world. Moreover, obtaining reliable labels for traffic events is challenging [56]. For these reasons, the focus of this thesis is an entirely unsupervised setting (without any training labels).

[1] An open-source network intrusion prevention and detection system, at <www.snort.org>

[2] Suricata is a free and open-source, mature, fast and robust network threat detection engine, at <https://suricata-ids.org/>

Existing research on unsupervised traffic anomaly detection is affected by some critical limitations: existing works either make strong assumptions to identify specific threats (e.g., similar communication patterns for botnet identification [21]), or require unencrypted traffic (e.g., [5, 36]). Other methods that work even in the presence of encrypted traffic either assume specific threats (e.g., data exfiltration [39]) or do not scale to large networks (e.g., IoT traffic of surveillance cameras [41]).

In this thesis, we propose MINOS[3], a *fully unsupervised* approach that produces an anomaly score for each internal host of an organization. That anomaly score can prioritize and classify (in an unsupervised manner) each host into one of three categories: clean, under attack, and infected. Also, it can recognize the attack time of an infected or under attack host. The inputs of MINOS are network communications between the internal hosts of an organization and the Internet, where no ground truth is required. To address the heterogeneity of network communications, MINOS automatically creates a behavioral traffic profile for each host independently by clustering network flows. We remark that MINOS is a fully unsupervised approach followed by a parallel procedure over multiple hosts. We experimented MINOS over 1,000 hosts and one year of network traffic at Boise State University. We then evaluated how the anomaly score of MINOS prioritizes and differentiates three classes of hosts effectively. We also show that MINOS has better accuracy and execution time

[3] In Dante Alighieri's Divine Comedy, MINOS is depicted as a man with a serpent tail in charge of judging evil souls to determine which circle of Hell they deserve to be in. The circle is determined by the number of wraps of MINOS's tail on the evil soul.

than Kitsune [41], a state of the art approach for unsupervised traffic anomaly detection.

1.2 Contributions

In summary, this work makes the following main contributions:

- We propose a novel fully unsupervised large-scale traffic analysis approach, called MINOS, that is able to classify internal hosts into one of three classes: clean, malicious, and under attack and also identify the time frame of being attacked. This is different from most prior research that distinguished just between benign and malicious activities. We evaluate MINOS on one year of real data collected for 1,000 hosts of Boise State University (a large academic network).
- In addition to offline analysis of one year of traffic data, we show that MINOS retains high performance in classifying hosts even when applied on reduced time windows (e.g., two weeks).
- We show that MINOS outperforms Kitsune [41] (a state-of-the-art approach for unsupervised traffic anomaly detection) both in accuracy and in execution times.

Our results show that MINOS is a viable solution towards identifying risky hosts in large networks in the absence of label supervision.

1.3 Outlines

In **Chapter 2**, we provide the relevant background of Intrusion Detection Systems, machine learning in supervised and unsupervised contexts, neural networks, commonly used standard novelty detection algorithms, and present the literature review of previous research works related to this thesis.

In **Chapter 3**, we discuss the size and shape of the raw dataset, experimental machine selection, network flows collection, and feature selection procedure. Also, we present the problem statements of this thesis.

In **Chapter 4**, we describe the methodology and implementation of feature vector extraction and novelty detection algorithms, i.e. identifying host status and time frame of existing attacks.

In **Chapter 5**, we extend our discussion on the experimental setup, variants of experimental instances and statistical relevance.

Finally, in **Chapter 6**, we discuss a summary of the proposed methodology, the future research direction and conclude this thesis.

CHAPTER TWO: BACKGROUND AND RELATED WORK

This chapter introduces the relevant background on intrusion detection systems, machine learning, neural networks, novelty detection procedure, some related works in the literature and comparison of them with our hypotheses. For more details about machine learning and neural networks, we recommend readers to a book by Goodfellow et al. [20]. For novelty detection algorithms, a survey paper by Pimentel et al. [48] is recommended.

2.1 Intrusion Detection System (IDS)

IDS is a process of monitoring and identifying computer and network events to determine the evolution of any unusual incident, which is considered to be an intrusion [1]. Generally, it detects undesired exploitation to the computer system, both through the Internet and the Intranet.

For example, a thief is standing in front of an anonymous house, looking around, investigating the surroundings, and then starts turning the knob of the front door. Unfortunately, the door is locked, so he moves to a nearby window and smoothly tries to open it. Unluckily, that is locked too. It demonstrates that the house is safe. If the house is safe in this way, why do people install an alarm in their home? Similarly, the common question for intrusion detection researchers: why researchers bother detecting intrusions if they established firewalls, patched operating systems, and checked passwords for soundness? The most straightforward answer to this question is intrusions still happen.

However, firewalls contradict with IDS in the sense that they cannot usually search for anomalies or specific content patterns to the same degree as IDSs do. Moreover, unlike firewalls, IDSs are automated because they do not depend on a human decision. As such, people occasionally skip updating a firewall's rule set correctly as they sometimes forget to lock their window. Therefore, developing an IDS becomes worthy of discovering and reacting for any computer attacks [31].

2.2 Types of IDS

Figure 2.1 illustrates the taxonomy of IDS, which is reproduced from [59]. Generally, IDS can be divided into two basic categories based on their position in the network or audit source location:

- *Host-based IDS (HIDS)*
- *Network-based IDS (NIDS)*

NIDS can be divided into two categories based on the source of data to be analyzed in NIDS:

- *Packet-based NIDS*
- *Flow-based NIDS*

Also, depending on the detection model IDS can be classified into two categories:

- *Signature-based IDS (SIDS)*
- *Anomaly-based IDS (AIDS)*

•

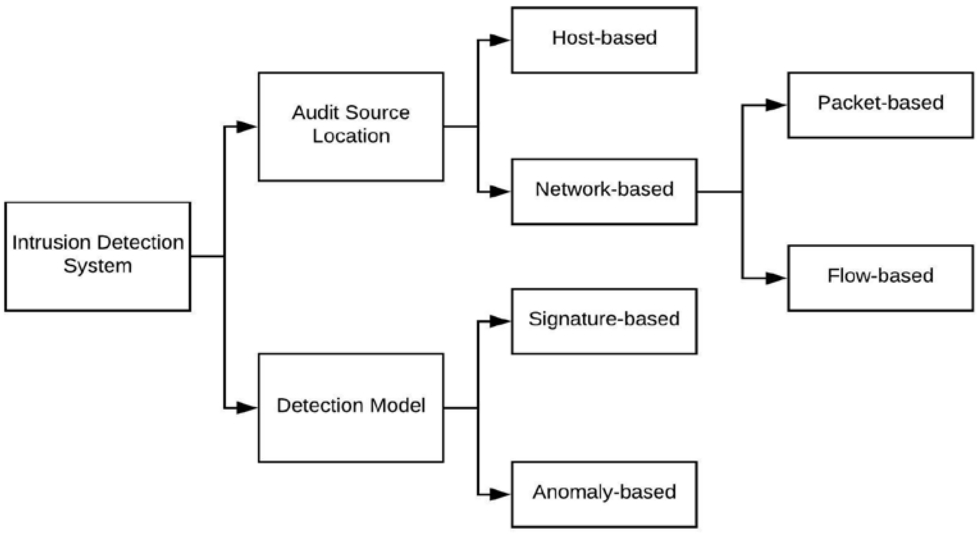

Figure 2.1 IDS Taxonomy

2.2.1 Host-based IDS (HIDS)

A HIDS is capable of monitoring a single machine and audit data (resource usage and system logs) traced by the hosting operating system [1]. It gives deep visibility of critical systems and refers to protect the environment by detecting and responding to malicious or anomalous activities. However, HIDS does not provide a complete picture of the security posture. HIDS log data needs to correlate with other critical security data and the latest real-world threat intelligence. In this context, HIDS seems like an agent that can monitor whether internal or external, anything or anyone, have blockaded the system’s security policy.

2.2.2 Network-based IDS (NIDS)

NIDS is used to monitor a network and analyze traffic to protect a system from network-based threats. Generally, a NIDS reads all inbound packets and searches for any suspicious patterns. If it can identify any risks, the system can take action by notifying administrators or blocking the source IP address from accessing the network [1]. As our goal is to identify a machine's status by scrutinizing network flows solely, we do not consider HIDS for this research.

Comparison between HIDS and NIDS: In contrast to HIDS, NIDS has some advantages. In NIDS, the deployment of a new host in the network does not need extra effort to monitor the network activity. Also, NIDS is less expensive because updating one component of NIDS is more comfortable than many components of HIDS on hosts. A NIDS presents extensive research of a corporate network via scans and probes. NIDS allows administrators to protect non-computer devices, such as firewalls, print servers, VPN concentrators, and routers. More importantly, NIDS gives us flexibility with multiple operating systems, devices, and protection against bandwidth floods and Denial of Service (DoS) attacks.

2.2.3 Signature-based IDS (SIDS)

SIDS, also referred to as "misused-based" or "rule-based", works similar to antivirus software [1]. SIDS monitors packets in the Network and compares them with pre-configured and pre-determined attack patterns known as signatures. If there is a successful match with the current input, an alert is prompted. A well-known tool of SIDS is Suricata (an open source IDS tool), which monitors networks by matching each packet it observes against a set of rules. A rule consists of the following:

- ***The action:*** Determines what happens when the signature matches

- ***The header:*** Defines the protocol, IP addresses, ports and direction of the rule
- ***The rule:*** Some options, which explain the specifics of the rule

The following is the appearance of a static machine's alert at Boise State University, which has been produced by the set of rules of Suricata (See table 2.1):

"10/12/2017-22:35:01.319011 [**] [1:2009582:3] ET SCAN NMAP -sS window 1024 [**] [Classification: Attempted Information Leak] [Priority: 2] TCP 150.255.174.211:61512 -> 132.178.137.210:873"

When an alert happens, it is essential to figure out what it means. Is it severe, or relevant, or merely a false positive? To find out more about the alert produced by Suricata, it is always a good idea to look at the category of the alerts, classification message, and priority of the alert. The alert mentioned above is in the category of *"ET SCAN"* rule, the classification message is *"Attempted Information Leak"*, and priority is *"2"*. Hence, *"ET"* indicates the rule came from the Emerging Threats project and *"SCAN"* indicates the purpose of the rule is to match on some form of scanning.

2.2.4 Anomaly-based IDS (AIDS)

An anomaly-based or behavior-based IDS can detect both network and computer intrusions by monitoring system activity and classifying it as either normal or abnormal. The classification is based on heuristics or rules, instead of patterns or signatures, and it attempts to detect any misuse or abnormal system operation. Figure 2.2 depicts some necessary modules for the general architecture of AIDS, such as parameterization, training, and detection. The parameterization involves accumulating raw data from a monitored environment, the training stage tries to model the system using manual or automatic methods, and the detection stage compares the system generated in the training stage with the elected parameterized data portion. Threshold criteria will be chosen to determine anomalous data [45]. Although different types of anomaly detection techniques are available, machine learning-based anomaly detection has become prominent. The overview of machine learning technique for AIDS is described in section 2.3.

Comparison between SIDS and AIDS: The advantages of SIDS are: ease of implementation, lightweight, low false-positive rates, and high true positive rates. One disadvantage, however, is its inability to detect any unknown attacks like AIDS.

2.2.5 Packet-based NIDS

In packet-based NIDS, all network packets that pass through a specific observation point are captured without any loss of information. For this reason, it is also known as *Deep Packet Inspection (DPI)*. Various observation points (i.e.,

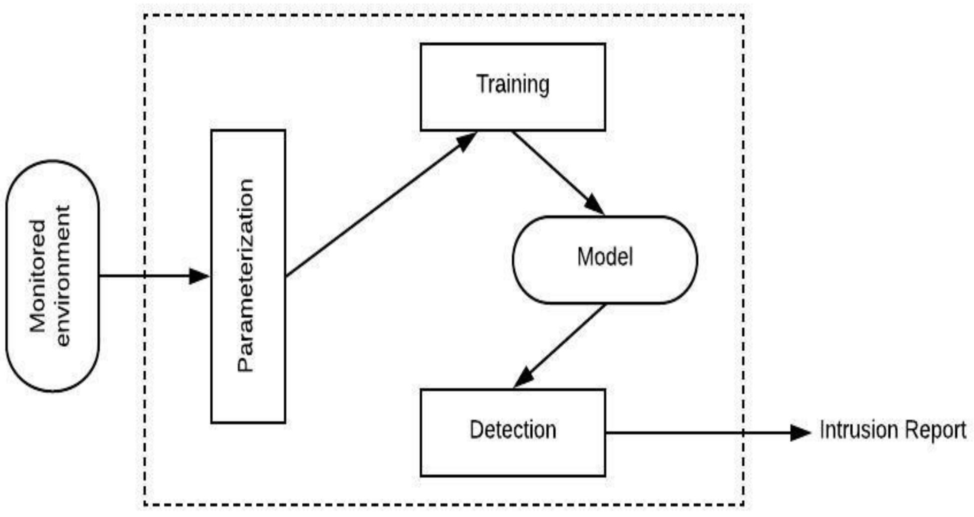

Figure 2.2 Generic AIDS functional architecture

routers, switches, network monitors, and so on) are dedicated to capture and analyze packets so that the resulting measurement data transfers to a remote analysis system.

A packet has two fields: the header (contains information about the source, destination, and others) and the payload (data). The packet-based NIDS scans these fields and determines whether or not a packet holds an intrusion. From the database, every single rule is checked against scanned incoming packets, as shown in figure 2.3. However, SIDS mostly uses a packet-based process.

2.2.6 Flow-based NIDS

The flow-based technique is a reputable data source in applications like network monitoring, traffic analysis, and security. Since flow data or network flow characterizes this method, flow-based NIDS is also referred to as “Network Behavior Analysis.”

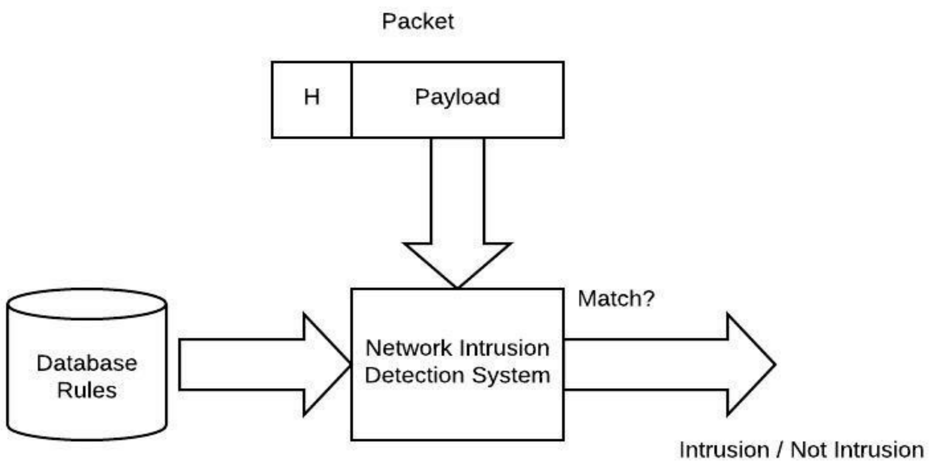

Figure 2.3 Packet-based NIDS

The definition of a flow can be *"a set of IP packets passing through an observation point in the network during a specific time interval, i.e., all packets belonging to a particular flow have a set of common properties"* [11]. Based on IP Flow Information Export (IPFIX) terminology [25], the common properties can be included with packet header fields (*flow keys*), source and destination IP addresses, source, and destination port numbers, protocol, and some meta information:

(ipSrc, ipDst, portSrc, portDst, proto)

The preparation and exportation format of flows are defined by two wellknown protocols: NetFlow and IPFIX. There are two components in a NetFlow setup: an exporter and a collector. The flow exporter can be a probe, a switch, or a router, which extracts the headers from each incoming packet noticed on the monitored interface. An exporter is responsible for creating the flow records from observed traffic and sending them over the network to the collector. The collector stores these flow records for further analysis and prepares them suitable for NIDS (figure 2.4). We have utilized one year of NetFlow data at Boise State University to develop an intrusion detection model that can prioritize and classify a set of machines.

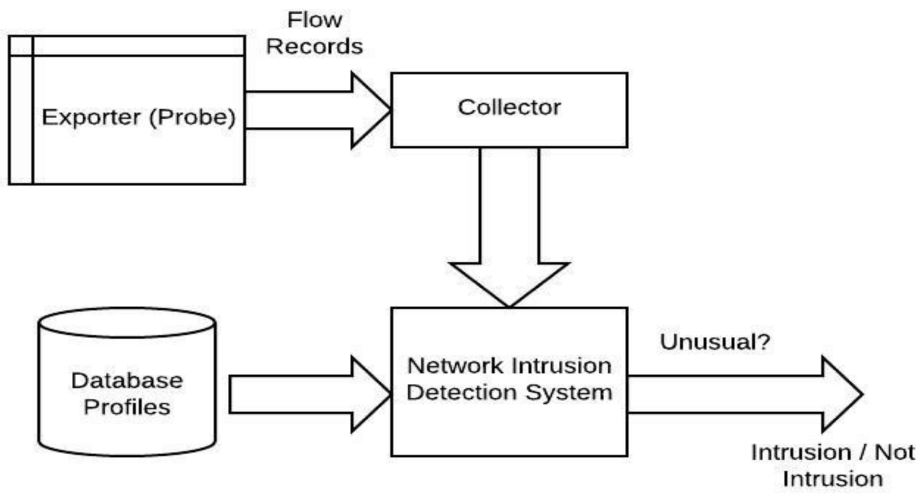

Figure 2.4 Flow-based NIDS

Comparison between Packet-based and Flow-based NIDS: The packetbased NIDS cannot detect any unknown attack as it compares only the predefined and known malicious signatures. Therefore, it is a highly resourceintensive task, expensive on a high-speed network, and infeasible in case of an encrypted payload. On the other hand, flow-based NIDS can handle considerably lower amount of data because it considers only the packet’s header field instead of its payload. For this reason, on a high-speed network, flow-based intrusion detection is more scalable than any of the other approaches. Moreover, flow exporters are widely deployable, meaning there is no need for additional capturing devices, and is less privacy-sensitive.

2.3 Machine Learning Technique (MLT)

MLT is a form of applied statistics, which emphasizes the use of computers to learn complex mathematical functions. To be more specific: "A computer program is said to learn from experience E concerning some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [42].

One of the practical applications of MLT is to comply with next-generation IDS, because it can build the required model automatically by depending on a given training dataset, which can be expressed using a set of attributes (features) and associated labels. The features can be of different types: categorical or continuous [45], and they are responsible for the applicability of anomaly detection technique. On the other hand, the labels associated with data instances are usually in the form of binary values, i.e., normal and abnormal. The favorability of this technique is linked to the availability of the essential training data. MLT classifies into supervised and unsupervised anomaly detection algorithms based on the nature of the dataset where they are originated and learned.

2.3.1 Supervised Anomaly Detection

Supervised methods or classification methods require a labeled training dataset containing both normal and abnormal samples to construct a predictive model.

Each training example consists of the independent variable(s) defining the input domain of data and the dependent variable(s) representing the target. Given a set of N training examples of the form $\{(x_1, y_1), ...(x_N, y_N)\}$ such that x_i is the feature vector of the i^{th} example and y_i is its label. A supervised learning algorithm can be formulated as authorizing a computer to learn a function $f : X \rightarrow y$, where X represents the space of independent variables (input space), and y is the space of dependent variables (output space).

Although supervised methods have a better detection rate than semi-supervised and unsupervised approaches, some technical issues were obtained; they are not as accurate as they are thought to be. For example, the deficit of a training dataset hinders its ability to achieve correct labels and noises of the training set will cause high false alarm rates. In the literature, the most common supervised algorithms are Supervised Neural Networks, Support Vector Machines(SVM), k-Nearest Neighbors, Bayesian Networks, and Decision Tree [45].

2.3.2 Unsupervised Anomaly Detection

Unlike supervised techniques, unsupervised anomaly detection techniques do not need any training data. They depend on two underlying assumptions, (1) most of the network connections are normal traffic, including a very little abnormal traffic, and (2) malicious traffic is statistically variant behavior from normal traffic. According to these assumptions, frequently appearing data groups of similar instances are assumed to be normal traffic, while infrequent examples are considered to be malicious. The most common unsupervised algorithms are clustering, anomaly detection, and novelty detection algorithms.

2.4 Novelty Detection

Novelty detection is a machine learning task for identifying new or unknown data. Recognizing abnormal system behaviors with the normal state of a system is the goal of novelty detection [40]. The system learns a model of the normal environment that does not have any malicious activities, and instead of finding any faults, the novelty filter detects the anomaly score deviations from this model. Although unique events occur infrequently, it can have significant consequences to overall system operation [10].

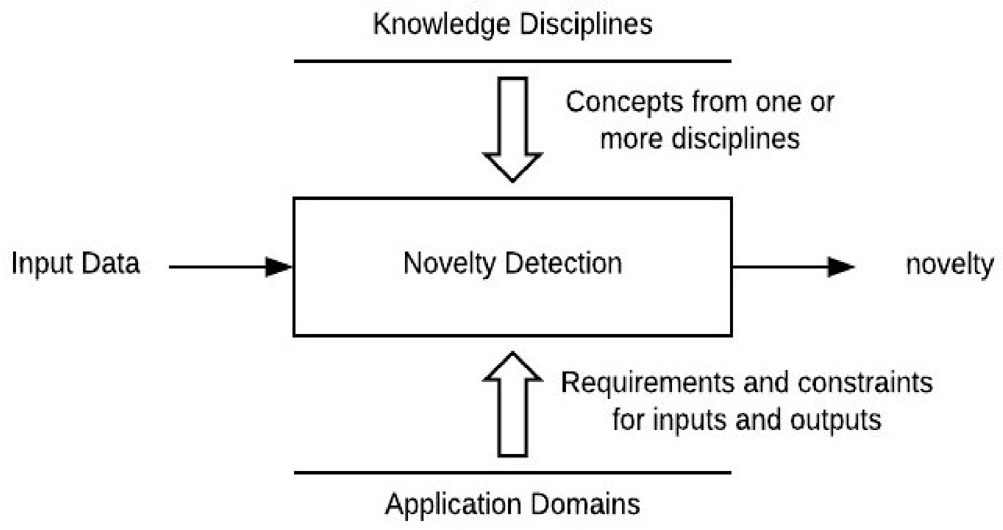
Knowledge Disciplines
Concepts from one or more disciplines
Input Data
Novelty Detection
novelty
Requirements and constraints for inputs and outputs
Application Domains

Figure 2.5 Framework of novelty detection

Figure 2.5 represents a general framework design of novelty detection, a combination of knowledge disciplines and application domains. Hence, knowledge discipline refers to several mathematical and algorithmic concepts, where application domains follow system expertise.

Input data in the novelty detection framework passes through several phases: preprocessing (remove an artifact from the data), feature extraction (input signals using a comparatively smaller volume), and construction of feature vectors followed by normalization (component-wise normalization). Afterward, the novelty detection method accepts the obtained feature vectors as input and delivers the information about the novelty as an output. Figure 2.6 illustrates the normal and abnormal data in feature space.

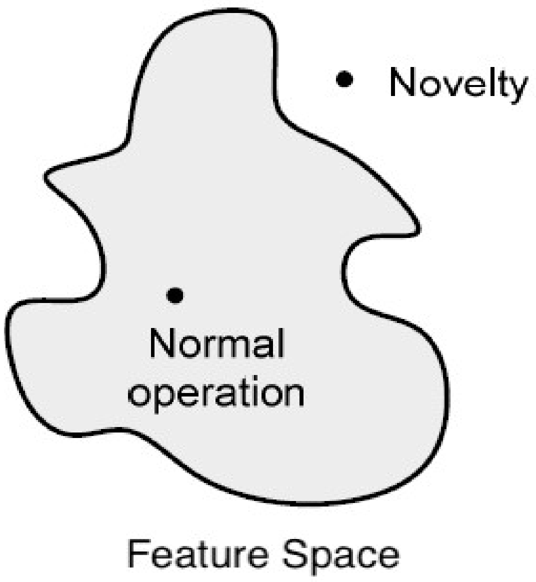

Figure 2.6 Novelty within feature space

Novelty detection has extensive applications in fields involving large datasets generated from critical systems. These include: cyber intrusion detection [18, 49, 64], terrorist activity, system breakdown, fraud detection [6, 16, 47, 61], data leakage prevention [55], electronic IT security [23], healthcare informatics, medical diagnostics, industrial monitoring and damage detection, image processing, video surveillance, text mining, sensor networks [8, 48], and many other specialized applications [19].

Also, novelty detection has been extensively applied to detect any new attack in IDSs, which falls within the application of novelty detection algorithms. However, identifying a machine attack status using the novelty detection algorithms in network flow datasets is the primary goal of this research.

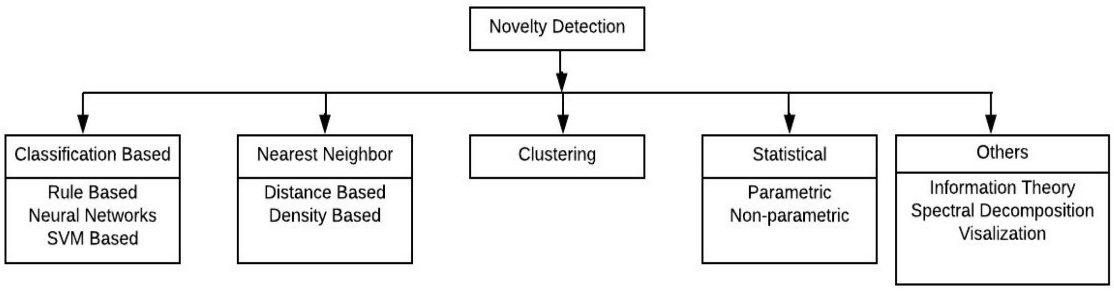

Figure 2.7 Taxonomy of novelty detection methods

Taxonomy of novelty detection is presented in Figure 2.7. We have applied K-means clustering to create the input feature vector for novelty detection methods. Similarly, clustering-based novelty detection techniques: Gaussian Mixture Model (GMM), One-class Support Vector Machine (OC-SVM), and reconstruction-based (neural networks, autoencoder) algorithms are used to compute the anomaly score. Based on this anomaly score, we have identified the attack status of each machine in the network.

2.4.1 K-means algorithm

Let $X = x_i$, $i = \{1, ..., n\}$ be the set of n-dimensional points to be clustered into a set of K clusters, $C = C_k, k = \{1, ..., K\}$. A K-means algorithm decides a partition in such a way that the squared error between the observed mean of a cluster and the points in the cluster is minimized. Let μ_k be the mean of cluster C_k. The squared error between μ_k and the points in cluster C_k is defined as:

$$J(C_k) = \sum_{x} i \in Ck \; ||x_i - \mu k||^2$$

The goal of K-means is to minimize the sum of the squared error for K clusters,

$$J(C_k) = \sum_{k=1}^{K} \sum_{x_i \in C_k} ||x_i - \mu_k||^2$$

Minimizing this objective function is known as an NP-hard problem, even for K = 2 [27]. Thus K-means, a greedy algorithm, only concentrates on a local minimum. Generally, K-means starts with a primary partition with K clusters and selects patterns of clusters to reduce the squared error. Since the squared error is always inversely proportional to the number of clusters K (with J(C) = 0 when K = n), it can be minimized only for a fixed number of clusters.

The K-means algorithm depends upon three user-specified parameters: the number of clusters K (a critical parameter), cluster initialization, and distance metric. No perfect mathematical criterion exists to choose a value of K, though some heuristics are available. K-means algorithms run independently with the distinct values of K, and builds the most meaningful partition in the domain. However, due to the convergence to local minima, different initializations can lead to inconsistent clustering. One approach for making the k-means method more efficient is to run the algorithm for a given K along with multiple different initial partitions and choosing the partition that refers to the smallest squared error. Another approach is to apply a filtering procedure that uses a spatial hierarchical dataset index while computing means, which also saves on costs. A third approach, explores the micro-clustering idea, which first groups the nearby objects into “microclusters” and then performs k-means clustering on the microclusters [22, 35].

2.4.2 Gaussian Mixture Model (GMM)

GMM is a non-bayesian, parametric probability density-based model, that uses fewer kernels than the number of patterns in the training set [40] to estimate the frequency. GMM uses optimization algorithms: conjugate gradients or reestimation techniques such as the Expectation-Maximization (EM) algorithm for fitting the training data. They are followed by maximizing the log-likelihood of the training data to choose the parameters of the model.

2.4.3 One Class Support Vector Machine (OC-SVM)

The OC-SVM is an unsupervised learning method that is aware of only a single class of data. It distinguishes between vectors which are referred to as either in class (inside the trained distribution) or outliers (outside the distribution) and it lies between the origin and the optimal separating hyperplane. The output score of OC-SVM represents the distance from the data point being tested to the optimal hyperplane. Whereas, positive scores denote normal behavior (with higher values representing greater normality) and negative values describe abnormal behavior (with lower scores representing larger abnormality) [12].

Generally, for novelty detection, OC-SVM uses a kernel trick to construct a hyperplane to separate the normal data from the original with maximum margin in a feature space [52]. OC-SVM assumes that a few training data points fall into some regions and drives the regions to be small in a feature space associated with the kernel [13]. The kernel trick, complied by the OC-SVM, makes it simpler to separate the normal data from the origin in a higher dimensional feature space.

2.4.4 Autoencoder

An autoencoder is an artificial neural network, commonly used for unsupervised novelty detection, based on the reconstruction error of the training examples. An autoencoder is trained to copy its input to its output along with the common purpose of nonlinear dimensionality reduction, which is the process of lessening the number of random variables under consideration. An autoencoder has two architectural parts: encoder and decoder. The encoder function creates either single or multiple hidden layers, that contain a code to describe the input, whereas, the decoder produces a reconstruction of the input from the hidden layer. Having a hidden layer smaller than the input layer is beneficial as an autoencoder is forced to create a compressed representation of the data in the hidden layer. This representation facilitates the classification, visualization, communication, and storage of data [24].

However, the intuition of an autoencoder is to obtain a higher reconstruction error for the novel or unknown data. Autoencoders are also trained with only the known examples in training data. While the optimized embeddings are learned using the training data, the reconstruction errors are computed for both the known and novel data in the test datasets. Here, the reconstruction error is proportional to the chances of the data point to be unknown. Figure 2.8 shows a three layer autoencoder: an input layer (Layer 1), one hidden layer (Layer 2), and an output layer (Layer 3), where the hidden layer captures the

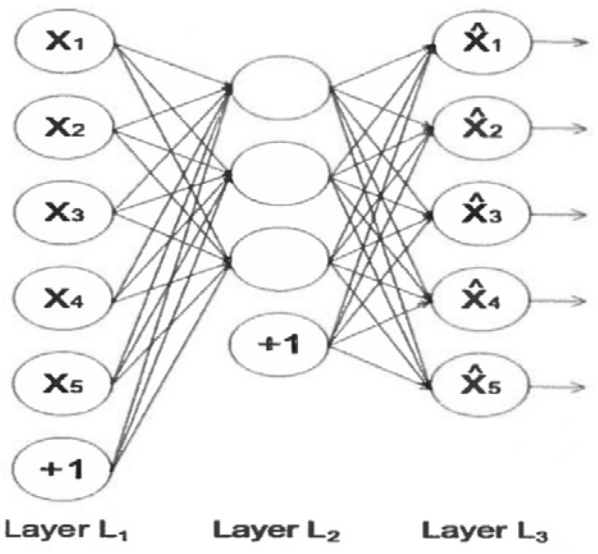

Figure 2.8 Simple architecture of an autoencoder

embeddings of the input layer into lower-dimensional space. These embeddings are used by the output layer to reconstruct the original data [50].

2.4.5 LSTM Autoencoder

Generally, humans do not start thinking from scratch in every second. For example, readers understand each word based on their perception of the previous words. They do not drop everything each time and start rethinking from the beginning; their thoughts have persistence — the same procedure followed by the recurrent neural network, which allows information to persist. The Recurrent Neural Network (RNN) allows forward and backward connections between neurons. Long short-term memory (LSTM) is an example of RNN architecture that recognizes values over random intervals. Stored values are not modified as learning proceeds.

Implementing an autoencoder for sequence data by using an Encoder-Decoder LSTM architecture is called an LSTM Autoencoder. An encoder-decoder LSTM is configured to read the given dataset of an input sequence, encode it, decode it, and then recreate it. The model's ability to recreate the input sequence represents the performance of the model. While the model obtains a desired level of performance by recreating the sequence, the model may drop the decoder part, leaving only the encoder model. Afterward, this model can be used to encode input sequences to a fixed-length vector. The resulting vectors are not limited as a compressed representation of the sequence or as an input to another supervised learning model, and preferably it can be used in a diversity of applications.

2.5 Performance Metrics

Similar to other machine learning algorithms, it is crucial to evaluate the performance of our novelty detection algorithms. Therefore, we calculated novelty scores for the flows dataset of each machine, and then used these scores and the ground-truth to compute the Area Under Receiver Operating Characteristic (AUROC) and Average Precision (AP) scores. We have described the ground truth collection procedure to evaluate our experimental result in Chapter 3.

2.5.1 Area Under Receiver Operating Characteristic (AUROC)

AUROC score computes the discriminating ability of classifiers or novelty detection algorithms to correctly classify a dataset into different categories such as known category or novel category. In our approach, we can understand how well the AUROC score distinguishes between good machines with the machines under attack or malicious machines. The AUROC is a plot with a false positive rate (incorrectly identifying as being anomalies) of a discriminating model as the x-axis and a true positive rate (correctly identifying as being anomalies) in the y-axis.

2.5.2 Average Precision (AP)

AP is more commonly averaged over all queries and reported as a single score, which characterizes as a prominent performance measure in the novelty detection domain. AP score reports the ability of novelty detection algorithms to distinguish different objects as novel. In real work scenarios, it is common to have a tiny proportion of new samples compared to the ordinary case. Therefore, our goal is to prove the novelty detection algorithms are suitable to discriminate the relatively small population of new examples. Hence, the false-negative rate is pretty dangerous for discriminating models like novelty detection.

2.6 Suricata Rule Category

We considered one-year of traffic flow and collected recorded alerts from a Suricata log file as the ground truth for this research. Suricata is an open-source, free, mature, fast, and robust network threat detection engine. It is capable of real-time IDS, inline intrusion prevention (IPS), network security monitoring (NSM), and offline pcap processing. The network traffic using powerful and extensive rules and signature language and has powerful Lua scripting support for the detection of complex threats. Most importantly, Suricata’s fast-paced, community-driven development focuses on security, usability, and efficiency.

We did not use ground truth for the input of our experiment, but we did use it to evaluate our experimental results. Regarding static machines, we found several alerts based on the ruleset of Suricata. Table 2.1 clearly describes the rule set category at Boise State University, which has been producing various alerts.

Based on some rules, Suricata can produce false positives. In that case, humans need to start with only a few rules and work their way up. Otherwise, it just gets overwhelming. On the other hand, if humans failed to set an important attack detection rule, there is a high chance to have a false negative. In this context, Suricata depends on personal decision or rule set-up. In this research, we found a total of 25 rule sets into the Suricata log file at Boise State University, which were decided by the security analyst at Boise State University. However, for this research, we do not need to know which alerts are being produced by Suricata. Instead, we only need the classification of machines in the network.

Table 2.1 Description of rule set of Suricata

#	Rule Category	Description
1	SCAN	Early warning can detect and identify host and network vulnerabilities in our environment. Scans can perform external attack simulations and comprehensive vulnerability checks along with registry evaluation.
2	POLICY	Application Identification category, which includes signatures for applications like DropBox, Google Apps, among others and also covers off port protocols. This alert is saying "I saw unencrypted HTTP traffic traveling over a port generally reserved for HTTPS encrypted traffic".
3	DOS	A cyber-attack in which a legitimate user is unable to access information systems, devices, or other network resources because of the actions of a malicious cyber threat actor.
4	COMPROMISED	A collection of known compromised machines, confirmed and regularly updated. This list waved from a hundred to several hundred rules based on the data sources. Most importantly, Snort does not handle IP matches well load-wise. Therefore, if your sensor has already pushed to the limits, this set would add a significant load.
5	CINS	Collective Intelligence Network Security (CINS) is a network of "sentinel" machines running around the internet, which allow a company to monitor those attacks leveraged against them and score them appropriately.
6	DROP	An IP based attack for some rules to block Spamhaus “drop” listed networks and daily updated collection of the Spamhaus DROP (Do not Route or Peer) list. Primarily it is known as professional spammers.
7	INFO	General rules to track suspicious host network traf-fic.

8	TOR	An IP based rules for the identification of traffic from and to TOR exit nodes.
9	P2P	Rules for the identification of Peer-to-Peer traffic and attack against, including torrents, edonkey, Bittorent, etc.
10	DNS	Rules for attacks and vulnerabilities for DNS besides the category for abuse of the service such as tunneling.
11	SNMP	Rules for attacks and vulnerabilities regarding the Simple Network Management Protocol.
12	WEB-SERVER	Rules for attacks and vulnerabilities against web servers.
13	MALWARE	Related to Malware and Spyware, where there is no clear criminal intent present. The threshold for formation in this set is typically some form of tracking, which stops short of apparent criminal activity.
14	EXPLOIT	Rules to detect direct exploits such as Windows exploit, veritas, etc.
15	GAMES	Rules for the identification of gaming traffic and attacks against games.
16	CHAT	Identification of traffic-related to various chat clients, irc and possible check-in activity.
17	USER-AGENT	Rules for identification and detection of user agent.
18	VOIP	Rules for attacks and vulnerabilities against the VOIP environment.

19	TFTP	Rules for attacks and vulnerabilities regarding the TFTP service.
20	FTP	Rules for attacks and vulnerabilities regarding the FTP service.
21	SCADA	Rules for the signatures of SCADA attacks, exploits and vulnerabilities.
22	MOBILE-MALWARE	Rules for the specification of mobile platforms such as malware and spyware related.
23	CURRENT-EVENTS	Rules for active and short-lived campaigns, which covers exploit kits and malware that will be aged and removed instantly due to the temporary nature of the threat.
24	SHELLCODE	Dedicated to Remote Shellcode detection. Remote Shellcode is used while an intruder wants to target a vulnerable process that is running on a different machine on a local network or intranet. After successful execution, the shellcode grants the attacker access to the target machine across the system.
25	TROJAN	Highly significant ruleset that can detect malicious software, which has an apparent criminal purpose. Rules discover malicious software that is in transit, active, infecting, attacking, updating, and whatever else Suricata can identify on the wire.

2.7 Related Work

There are two significant approaches to detect malicious activities in network traffic: *misuse-based* and *anomaly-based*. Misuse-based systems rely on manually defined signatures which embed expert knowledge of a priori known attacks. Such systems are commonly deployed in large enterprises through NIDS that perform online DPI on traffic to detect whether a packet (or a set of packets) matches one of the detection signatures. Some examples of NIDS are Snort, Suricata, and Bro [46]. However, such systems do not scale well with the increasing number of activities and attacks in network traffic as the signatures require manual definition, and most importantly they are limited by requiring a priori knowledge of the attack scenarios [56].

Anomaly-based systems create behavioral models of traffic towards the goal of detecting malicious activities within a network, even in the absence of prior knowledge about attacks [9]. Hence, many research efforts have been focused on anomaly-based methods, but many challenges complicate the application of statistical methods to traffic more than other domains [56] (see also Section 3.7 in Chapter 3). Some existing approaches [7, 18, 29, 54] require labeled traffic datasets (which are hard to obtain and limit the efficacy of the approach to a priori attacks) or have too high computational times (which hinders applicability of both offline and online analysis of traffic). On the other hand, we propose a fully unsupervised approach that does not require any label, and our approach is highly efficient, requiring less than 2 minutes to process a half month of network traffic activities for 1,000 machines. Authors of [63] propose an OC-SVM based method trained on malicious traffic; however, their approach is supervised as it requires traffic labeled as malicious and does not generalize to unseen malicious traffic. On the other hand, we aim to create a behavioral model for every machine and identify which machines are more anomalous.

Deep Learning (DL) or ANN has been used extensively in the anomaly detection process, because of its capability to learn complex concepts and the concepts from the domain of network communication. There have been also many efforts in applying DL [7] , ANNs [43, 44, 60, 66] and AutoEncoders [28, 65] for traffic anomaly detection. However, such models are supervised and require periodic re-training, which is infeasible in modern high-speed networks and at the rate to which new attacks are appearing [56]. In contrast, MINOS can identify malicious, under attack and clean machines as well as the time frame of being attacked for a machine in an unsupervised way by lightweight analysis of network traffic flows.

The most recent and related work to our research is Kitsune [41], an unsupervised approach for online traffic anomaly detection. Kitsune relies on an ensemble of Autoencoders and takes as input packet-based features. Kitsune was designed for online analysis and has been evaluated on about two hours of traffic of IoT cameras, showing good results in detecting IoT botnet activities. We argue that two hours of traffic are reasonable to create a behavioral model of an IoT device, but it would not be enough to model client and server hosts of large organizations, where patterns are highly variable and may follow weekly and monthly patterns. Hence, by design, MINOS aims to create a behavioral model over more extended periods and uses NetFlow information using packet-based analysis that would be computationally infeasible to process for more extended periods. Nevertheless, as described in more detail in Chapter 5, we also try to adapt the AutoEncoder architecture used in Kitsune to our scenario, and we experimentally demonstrate that MINOS performs better than Kitsune both in terms of *detection capability* and *execution time*. Moreover, unlike Kitsune, MINOS can distinguish hosts into three classes, offering more intelligence to security operators, which also allows hosts that are not yet infected but under attack to take proactive steps.

CHAPTER 3: DATASET AND PROBLEM STATEMENT

In this chapter, we provide a short overview of the dataset for MINOS: raw dataset collection, preprocessing, experimental static machines selection, and identification of useful features. We also describe research challenges and the problem statement of this thesis.

3.1 Category of Data Sources

A few data sources have been utilized in NIDS, which do have the following properties [53]:

- **Scalability:** The capability of dealing with gigabyte networks
- **Lightweight:** Small size of obtained network data
- **Privacy:** Owing to the severe consequence of network data monitoring

Network data sources can be extensively characterized by the following categories:

- **The protocol-based data sources:** Protocol-based datasets comprises of Simple Network Management Protocol (SNMP) and Internet Control Message Protocol (ICMP).
- **The packet-based data sources:** In the packet-based approach, whole network packets have been used, and recognition is usually performed by the use of software such as tcpdump.
- **The flow-based data sources:** The flow-based approach is characterized by the use of network flows. According to literature,

> the flow is a unidirectional data stream between two machines where all transmitted packets of this stream share some common characteristics (source and destination IP addresses, source and destination port numbers, and protocol) [1, 59].

For the time being, a unique measurement system can provide some extra features with the general features of flow: the number of packets and bytes, the start and end time, the timestamp of first seen, and a TCP flag. NetFlow and IPFIX are two conventional protocols that define the preparation and exportation form of flows [25], which is known as *flow record*. We explained in figure 2.4 in chapter 2, how NetFlow exports flow in our network and makes it suitable for NIDSs for further analysis.

It is convenient for NetFlow to deploy network communications because almost all Cisco devices support at least one version of NetFlow. A file transfer that involved transferring gigabytes of data in high-speed networks characterizes as a comparatively small network flow. This flow builds up only a portion of the original file transfer such that the overhead caused by creating flow records is justified (the cost on account of NetFlow is in average 0.2 % [59]).

However, sampling techniques or flow aggregations can improve the performance of routers and monitoring stations [57].

3.2 Size of the Dataset

The dataset was obtained from the university network after monitoring their IDS's setup, which is used to track inbound/outbound traffic for specific segments. There was a 10-Gbps optical internet connection with a peak of 4.2 Gbps. They collected flows using Netflow version 5. This version extracts flow in the following ways:

- **Internet:** Inside the interface of the border firewall, which is responsible for providing the outbound or uploaded traffic.
- **Internetout:** Outside the interface of the border firewall, which is responsible for providing the inbound or downloaded traffic.

In 5 minute intervals, *nfdump* has produced 2400 to 10,500 flows per second. We selected 4*th* February 2017 to 4*th* February 2018 flow set for our research purpose. The total size of this unzipped data set is 6.5 *TB* along with 57.87 Billions of flows. We have parsed this massive dataset and chosen a partial amount of flows for this research.

3.3 MINOS Overview and Dataset

For the dataset collection, instead of using traffic benchmarks which have been shown to contain artifacts [37], we collected one year of real traffic for 1,000 LAN hosts at Boise State University[4]. To deal with the number of communications, we decided to use *network flows* (NetFlows) [39], which collect highlevel communications between any two hosts. NetFlow allows MINOS to be resilient to encrypted and obfuscated communications as no payload information is used.

MINOS operates in the following way. It takes as input network flows from a large organization, which can correspond to multiple hosts, and analyzes them in a fully unsupervised way (i.e., without any expert knowledge nor ground truth given

[4] Unfortunately, obtaining access to a real enterprise network traffic is almost impossible due to privacy concerns.

as input). Instead of labeling individual events, MINOS aims to classify the state of each host in the network as either:

- **Clean Host** (CH): A host not involved in any malicious activity.
- **Host Under Attack** (HUA): A host that is receiving attacks from the outside.
- **Malicious Host** (MH): An infected host that is performing an attack to the Internet from the internal network of the organization.

We point out that this novel separation into three classes is different from prior literature, in which only two classes of hosts were considered: malicious and benign [5, 39, 41]. To deal with the heterogeneity of activities among different hosts, MINOS automatically creates a personalized behavioral traffic profile for each host independently. Afterward, novelty detection is applied to obtain an anomaly score; the scores of all hosts are used as input for an unsupervised classification module that has associated one of the three classes to each host.

Table 3.1 Dataset details. It corresponds to one year of traffic collected at Boise State University (Feb 4th, 2017—Feb 4th, 2018).

Category	Num. Hosts	Num. Flows	Size(GB)
Clean Host (CH)	530	34,413,822	1.69
Host Under Attack (HUA)	437	35,022,898	1.7
Malicious Host (MH)	33	4,146,507	0.21
Total	1,000	73,583,227	3.6

A detailed breakdown of the dataset considered in this thesis is reported in Table 3.1. All 1,000 hosts are LAN hosts where the IP is assigned statistically. It is essential that the mapping between user and host be static so that MINOS can create a behavioral profile over time. As *ground truth* for evaluating our method, we have deployed a Suricata IDS and used the following criteria: a host h_i is *clean* if h_i does not generate any alert, it is *under attack* if h_i is in the destination IP of identified alerts (but h_i is never the source), and it is *malicious* if there is at least one alert in which h_i is the source of at least one alert. The Suricata configuration has been optimized for the environment by security analysts. Suricata alerts correspond to malicious activities such as botnet communications, command, and control interactions, exfiltration or exploit attempts malware drive-by downloads, and interactions with blacklisted external hosts [46].

The uniform selection procedure of 1,000 hosts has been described in section 3.5. After machine selection, we aggregated all the flows associated with those machines from the whole dataset of a one-year duration. Ultimately, the size of the dataset of 1000 devices is 3.6 GB of 73,583,227 traffic flows. Seen on Table 3.1, HUA consists of the maximum number of flows compared with others, which is almost 48 percent (1.7 GB) of the entire traffic (3.6 GB).

TABLE 3.2: Flows breakdown based on protocol

Protocol	Number of Flows
UDP	5,840,211
ESP	1
TCP	59,513,927
GRE	413,192
IPv6	7,091
SCTP	68
ICMP	7,808,741

Total	73,583,227

There are a total of seven IP protocols that exist in our dataset: User Datagram Protocol (UDP), Encapsulating Security Payload (ESP), Transmission Control Protocol (TCP), Generic Routing Encapsulation (GRE), Internet Protocol Version 6 (IPv6), Stream Control Transmission Protocol (SCTP), and Internet Control Message Protocol (ICMP). Table 3.2 illustrates the maximum number of flows is related to TCP protocol (80.88 % of the entire traffic).

Although we define ground-truth based on a signature-based system, MINOS operates in a *fully unsupervised* fashion and uses minimal information of the net flows. Moreover, it does not perform any deep packet inspection, works in the presence of encrypted communications and obfuscated payloads. Also, MINOS can operate in such a way in which signature-based systems would not work and does not rely on any expert knowledge.

3.4 Feature Selection

In our dataset, a flow closely follows the Netflow version 5 and has the following form:

$$(Date, Fseen, Dton, Prot, Isrc, I_{dst}, Psrc, P_{dst}, P_{ckt}, Bte, Flag)$$

Table 3.3 ***Flow-based Features*** **extracted from each flow.**

#	**Feature**	**Description**
1	Weekend/Weekdays (D_{ate})	It is 1 if the flow started in the weekend and 0 otherwise.
2	First Seen (F_{seen})	Timestamp of the beginning of the flow.
3	Duration (D_{ton})	Duration of flow in milliseconds.
4	Protocol (P_{rot})	TCP/IP Protocol of the flow.
5	Source Port (P_{src})	Source port of the flow.
6	Destination Port (P_{dst})	Destination port of the flow.
7	Packet (P_{ckt})	Number of network packets transferred in the flow.
8	Bytes (B_{te})	Number of bytes transferred in the flow.
9	Incoming/Outgoing (*Flag*)	It is 1 if the traffic flow is originated from a host internal to the network and is going to a host external to the network (outgoing). It is 0 if the traffic is generated from an external host and is going to an internal host (incoming).

The unidirectional communication is identified by the source and destination IP addresses (I_{src} and I_{dst}), the operated ports (P_{src} and P_{dst}), and the protocol type (P_{rot}). The fields P_{ckt} and B_{te} give the total number of transmitted packets and bytes respectively. The TCP header flags are stored as a binary "OR" in all packets of the flow (*Flag*). We selected eight out of these eleven features for our experiment. Hence, IP addresses are removed because they have been anonymized, and instead of a TCP flag, we used a flag that maintains incoming or outgoing flow direction.

Table 3.3 shows all the elected features of our experiment, including symbol and description.

After finalizing features, it is necessary to complete the preparation of the dataset by eradicating the categorical data. Hence, categorical data are variables that contain label values rather than numeric values. For example, the protocol of the input dataset is the string value. Some machine learning algorithms can support categorical data directly. For example, a *Decision Tree* algorithm can learn directly from categorical data where no data transforming is required. On the other hand, many machine learning algorithms demand all numeric input and output variables, instead of any label data.

Table 3.4 Minimum and maximum values of data set features

#	**Feature**	**Minimum Value**	**Maximum Value**
1	*Date*	0	1
2	*Fseen*	0	86,399,998.0 (milliseconds)
3	*Dton*	0	312.5 (seconds)
4	*Prot*	1	7
5	*Psrc*	0	65,535
6	*Pdst*	0	65,535
7	*Pckt*	0	977,002
8	*Bte*	0	937,800,000
9	*Flag*	0	1

Therefore, we converted the time stamp of the first seen (F_{seen}) attribute to milliseconds, changed the date (D_{ate}) to binary one or zero, based on weekends and weekdays, respectively. *Protocol*, *Source Port*, *Destination Port* are categorical features which we converted to numerical through one-hot encoding with sparse representation. Afterward, we considered the maximum and minimum value of all features to get an idea about the deviation of those attributes (table 3.4). Since seven types of the protocol (table 3.2) exist in our dataset, the minimum and maximum value are one and seven respectively due to the label encoder. As a preprocessing operation, each flow feature is normalized across all the flows, so that the mean is 0 and the standard deviation is 1.

3.5 Machine Selection

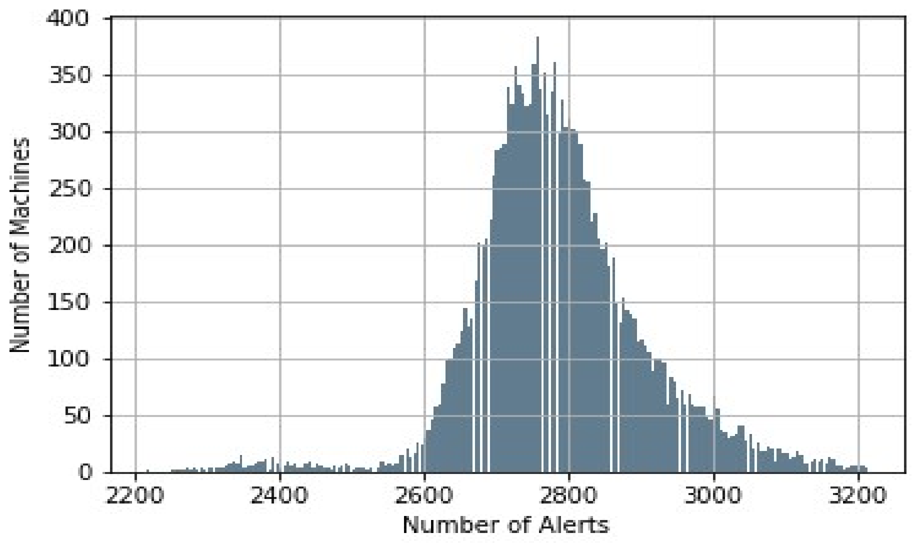

Figure 3.1 Number of machines vs. average number of alerts

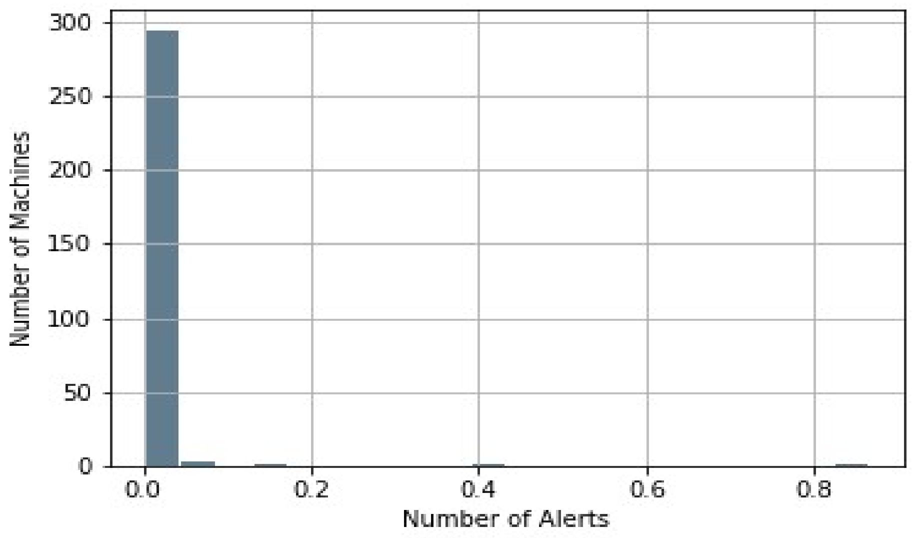

Figure 3.2 Number of machines vs. large number of alerts

At first, we collected all the static machine's list from network flow, and there were a total of 16,290, whereas 1,000 static machines have been targeted to do research. To select 1,000 machines in those three categories, we followed a procedure to create a balanced machine set. We selected 47% of 1000 = 470 as malicious or under attack machines, and 53% of 1000 = 530 as clean machines. Among 470 machines, 33 were malicious machines, and 437 were under attack machines. Between malicious and under attack machines, we picked 10% of 470 = 47 machines that contains a large number of alerts (48,000 — 8,661,605), and the rest of the 423 (470-47) machines have an average number of alerts (2600 — 3100). Figure 3.2 and 3.1 portrays this machine selection procedure. Hence, the number of flows were also in our consideration when we selected machines. Figure 3.4 and 3.3 represents these histograms. In synopsis, we selected 47 machines, where the number of flows lies between 200,000 and 375,000, and alerts 48,000 to 8,661,605, and the rest of the machines (423) flows in the range of 58,000 to 68,000 and the number of alerts are from 2600 to 3100. On the other hand, similar to malicious machines, those machines (530) have been selected as CH, whose number of flows lie between 58,000 and 68,000. Figure 3.5 represents this through a histogram. The reason behind this selection procedure is to assure that our experiment result is not biased with the number of flows.

3.6 Ground Truth Extraction

In the machine learning domain, the term "ground truth" indicates the accuracy of the training set’s classification. In other words, we can say this term is checking the results of a machine learning algorithm for precision against the real world. This term is borrowed from meteorology, where "ground truth" refers to

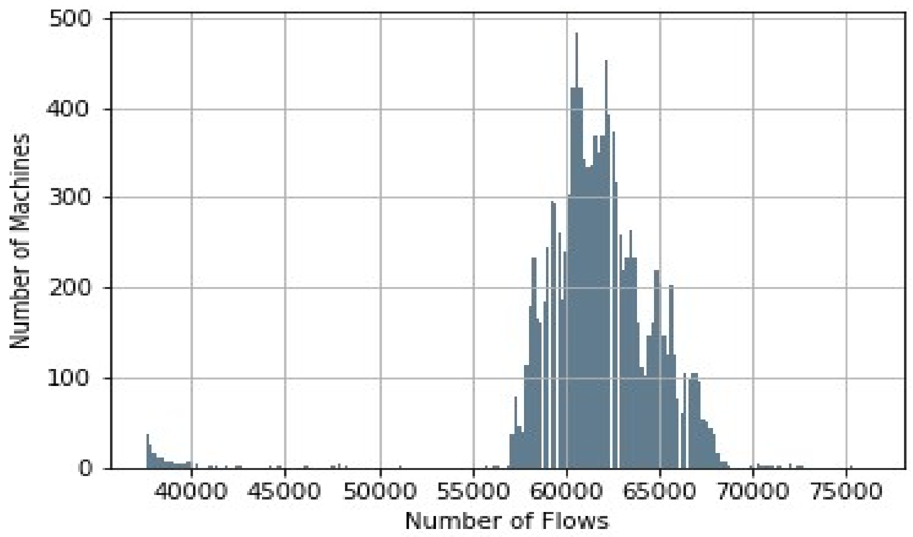

Figure 3.3 Number of Machines with average number of flows

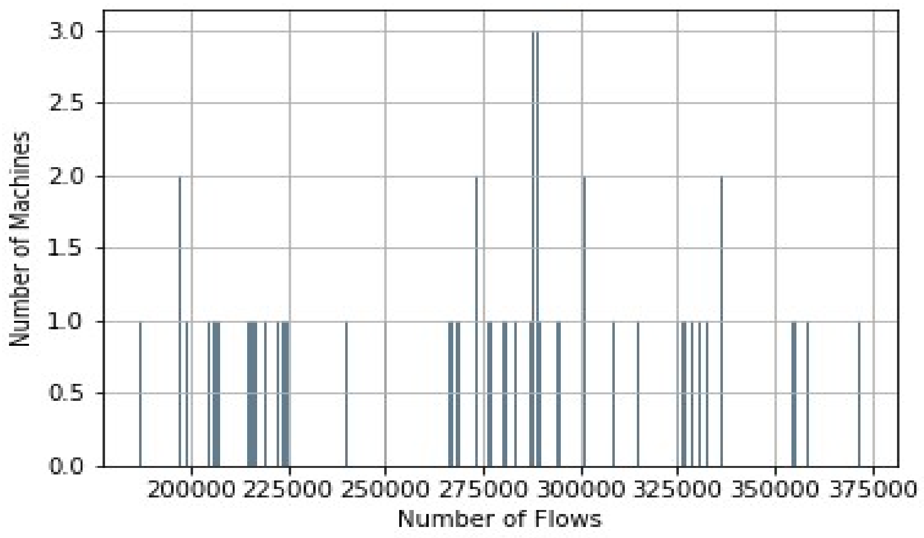

Figure 3.4 Number of Machines with large number of flows

information obtained on-site. Usually, the term implies a kind of reality check for machine learning algorithms and is used in statistical models concerned with proving or disproving research hypotheses.

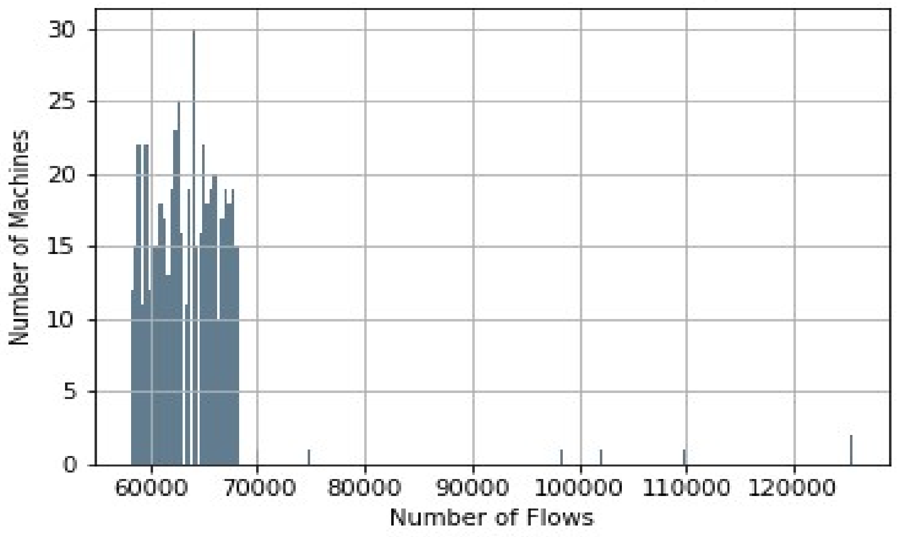

Figure 3.5 Number of Clean Machines with flows

Table 3.5 Ground Truth for a single machine

Timestamp for one hour	Number of alerts
02/04/2017/00:00:00.000000-02/04/2017/00:59:59.000000	0
02/04/2017/01:00:00.000000-02/04/2017/01:59:59.000000	0
02/04/2017/02:00:00.000000-02/04/2017/02:59:59.000000	5
02/04/2017/03:00:00.000000-02/04/2017/03:59:59.000000	0
⋮	⋮
02/04/2018/01:00:00.000000-02/04/2018/01:59:59.000000	10

We described three types of machines: CH, HUA, and MH and selected the type of machine based on the ground truth. MINOS is a fully unsupervised approach, and it does not need any labeling or prior knowledge. However, we collected ground truth from the log files of Suricata in order to evaluate our results. The Suricata log files from $4^{th}February$, 2017 to $4^{th}February$, 2018 were in our consideration as an input dataset belonging to this range. The following is the example of a single alert from the Suricata log file:

"10/12/2017-22:35:01.319011 [**] [1:2009582:3] ET SCAN NMAP -sS window 1024 [**] [Classification: Attempted Information Leak] [Priority: 2] TCP 150.255.174.211:61512 -> 132.178.137.210:873"

Into this alert the string ”$->$ ” divides two IP addresses and port numbers. The left side of that arrow (150.255.174.211:61512) indicates as source IP

(150.255.174.211) and source port number (61512). The right side of that arrow (132.178.137.210:873) means destination IP (132.178.137.210) and destination port number (873).

Let’s assume, "132.178.137.210" is a machine that is under attack, which exists in the alert as a destination IP address. There is also the exist date (10/12/2017) and the timestamp (22:35:01.319011) of that alert. We created a dictionary to collect the ground truth against a single machine. The key value of this dictionary is a particular hour against a specific date, and the value is the number of alerts that exist in that specific hour. Table 3.5 illustrated a sample of an identical machine’s ground truth, where the first, second, and fourth hour of $4^{th}February$ 2017 do not contain any alerts, whereas, in the third and last hour it holds five and ten alerts, respectively.

The problem with Suricata is that it is unable to detect any unknown attacks because it is a signature-based or rule-based anomaly detection approach. Moreover, it is analyzing packets to produce alerts against an attack, which is inconvenient for today’s high-speed networks. Also, it is infeasible for encrypted packets. Therefore, we proposed a novelty detection approach using only traffic flows instead of packets so that it is able to detect any attacks from encrypted traffic.

3.7 Research Challenges

Misuse-based systems rely on manually defined pattern matching signatures (e.g., NIDS [46]). It cannot cope with the continuously evolving and growing variety of traffic activities and attacks in large networks [62]. Moreover, these methods require a priori knowledge of the attacks. Hence, statistical anomalybased methods have been investigated, but their adoption is hindered by several challenges [56].

Dataset Collection. The first intrinsic challenge is to obtain a representative dataset to evaluate a proposed method. Prior research efforts exist to build *benchmark datasets* for traffic anomaly detection (e.g., DARPA [33], KDDCUP [30]), but successive research [37] has demonstrated that such datasets contain artifacts associated with the artificial injection of attacks, or that they are not representative of the traffic of large real-world organizations.

Quantity of Communications. We focus on large organizations, which can have thousands of hosts and billions of Internet communications per day. In such a scenario, it is incredibly challenging to detect which specific actions are malicious. Also it is challenging in terms of computational perspective. We will show how an existing state-of-the-art approach for traffic anomaly detection becomes unusable when applied in our domain.

Encrypted Communications. Some traffic anomaly detection methods assume that the traffic is not encrypted [5, 36, 62]. It is important to develop a methodology that can work in the presence of encrypted traffic and by using minimal information about the communications so that it captures high-level behaviors [39].

High Cost of Errors. Both false positives and false negatives have a much higher cost than in other domains. False positives correspond to *false alarms* and can quickly overwhelm security analysts if they have to investigate reported incidents [56] manually. False negatives correspond to *missed attacks*—if even a single host gets silently infected, then the whole organization is at risk. To complicate the situation, the majority of network activities are benign. Hence, due to base rate fallacy [3], it is even harder to detect real network threats. Therefore, it is crucial to achieving very high performance in this domain.

Heterogeneity of Activities. Each host has very different individual behavior. An enterprise can have a Web server, file server, database servers, WiFi and LAN clients, where employees can have a wide range of possible usage profiles [2, 39]. This varies greatly depending on the applications deployed on each host.

3.8 Problem Statements

We have designed MINOS taking these challenges into account. The objective of this research is two-fold. Firstly, we propose MINOS is a novel approach for fully unsupervised large-scale traffic analysis, are can prioritize and classify internal hosts into one of three classes: clean, malicious, and under attack. The goal is to find out the anomaly score of each machine and prioritize a group of machines in a completely unsupervised way by analyzing only net flows. If that machine is harmful, then identifying that machine as a malicious or under an attack machine. In this thesis, the unique features of the novelty detection technique makes it feasible to identify the state of the machine with high accuracy and less execution time. Secondly, we figure out when the attack has happened for a machine if the machine is under an attack or malicious.

CHAPTER FOUR: METHODOLOGY

Methodology

The MINOS approach consists of the following steps (Figure 4.1):

1. Collection of the sequence *TF_i* of traffic flows during a particular time frame (2 weeks to 12 months) for a specific host *h_i* [5]
2. Extraction of hourly-based features dataset by using the traffic flows of host *h_i*. The hourly-based features dataset consists of rows with features [$f_1, \ldots, f_m$], the flows starting in each specific hour of the defined time frame (in Figure 4.1 the time frame contains *d* hours) for the host *h_i*.
3. Learning an anomaly hourly-based model from the hourly-based features dataset and retrieving the anomaly score *score*(*$hour_j$*) for each hour *$hour_j$*.
4. Normalization of all hourly scores and aggregation to achieve a unique and absolute score for each host *h_i*.
5. Given all the absolute anomaly scores for all the hosts, group the scores in three categories and classify (in an unsupervised way) each host in one of the three classes: CH, HUA, and MH.

[5] We observe that such time frame lengths are required to create a realistic behavioral model of a client/server host that may perform different operations depending on the time of the day, day of the week, and month of the year [39].

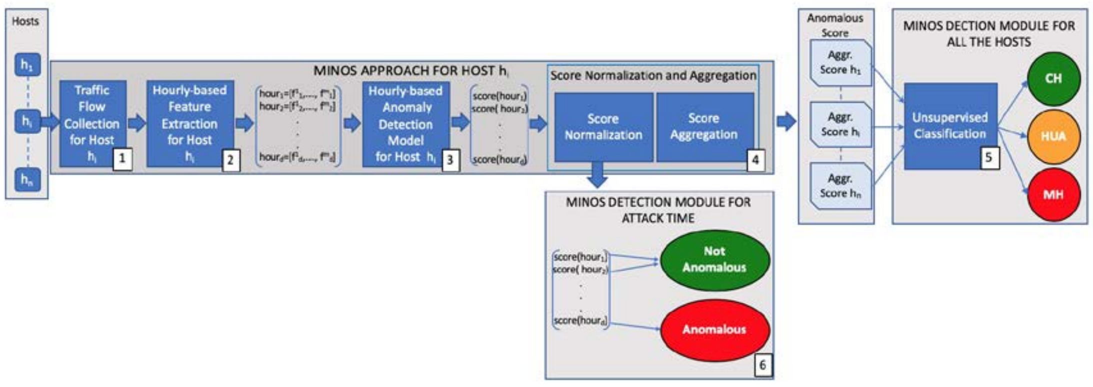
Hosts
MINOS APPROACH FOR HOST h_i
Traffic Flow Collection for Host h_i
Hourly-based Feature Extraction for Host h_i
Hourly-based Anomaly Detection Model for Host h_i
Score Normalization and Aggregation
Score Normalization
Score Aggregation
Anomalous Score
MINOS DECTION MODULE FOR ALL THE HOSTS
Unsupervised Classification
CH
HUA
MH
MINOS DETECTION MODULE FOR ATTACK TIME
Not Anomalous
Anomalous

Figure 4.1 The main steps of MINOS.

6. Identifying each hour as anomalous or not based on the anomaly score.

We use the aggregated anomaly score for each host to prioritize the dangerous machines and classify them in three categories (described in section 3.5 in Chapter 3): *Clean Host* (CH), *Host Under Attack* (HUA), and *Malicious Host* (MH). It is essential to notice that the MINOS procedure is fully unsupervised since it uses only traffic flow data as input. Moreover, the MINOS procedure is parallelizable for each host, depicted in Figure 4.1), where MINOS can quickly scale to analyze a large number of hosts. In the remainder of this chapter, we describe in details steps 2, 3, 4, 5, and 6 of MINOS.

4.1 Step 2: Hourly-based Feature Extraction

In this step, for each host, MINOS first extracts features for each flow (*flow-based features*), then processes them to obtain the *hourly-based features*. These hourlybased features summarize the behavior of the host across all hours of the specific time frame. The hourly-based features are the ones required to compute the anomaly score (Figure 4.1).

The hourly-based feature extraction step (Figure 4.2) takes an input of all the traffic flows within a specific time frame; then, it produces a set of features for each hour. Each flow $Flow_j$ is characterized by a timestamp t_j and a vector of flow features G_i. The features extracted from each flow are described in Table 3.3.

After computing G_i, we execute the K-Means clustering algorithms to obtain $\{C_1, . . . , C_k\}$ clusters. It is important to note that the *First Seen* feature (Table 3.3) is used to create clusters containing flows that are temporally close to each other. *Protocol*, *Source Port*, and *Destination Port* are categorical features which we convert to numerical through one-hot encoding with sparse representation. As a preprocessing operation, each flow feature is normalized across all the flows, so that the mean is 0, and the standard deviation is 1.

We used a machine-learning algorithm, $K - means$ to train a model for our input dataset. The motivation for choosing K-Means is that the number of flows for each host is large, and the K-Means clustering is the fastest clustering algorithm in terms of Euclidean space and similarity. The critical part is selecting the number of clusters because each machine contains a different quantity of flows. Hence, the static value of this parameter can be a hindrance for our desired outcome. For example, machine *A* has 58, 000 flows, and machine *B* has 5 *million* flows; if the static value of the cluster number is 50, it might be precise for machine *A* but not for machine *B*. Thus, we used the well-known rule of thumb [26] on choosing the best *k* for a $K - Means$ clustering: $k \approx \sqrt{\frac{g}{2}}$, where *g* is the number of points to the cluster. In our case, this *g* is equal to the cardinality of input flows matrix for each machine. Thus, the value of the cluster number, C_k, would be dynamic based on the length of the input flows matrix. Since the number of clusters, C_k, has to be an integer value, a ceiling operation has been done significantly over the rule of thumb.

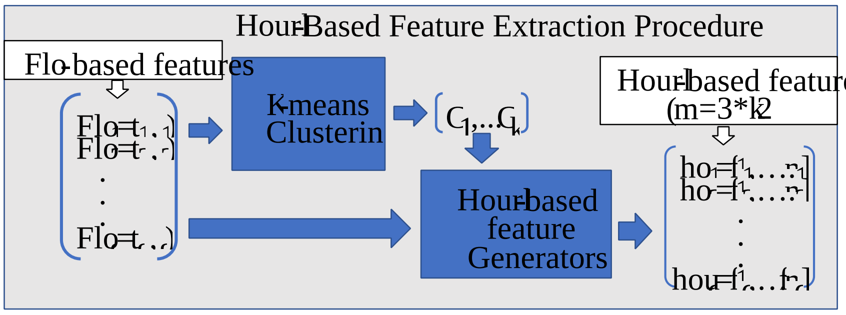
Hour-Based Feature Extraction Procedure
Flo-based features
K-means Clusterin
$C_1 \ldots C_k$
Hour-based features (m=3*k2
Hour-based feature Generators

Figure 4.2 MINOS extraction of *Hourly-based Features* for one host.

We now aim to define a set of features that summarize the behavior of a particular host for each hour. Let us introduce some elements: the clusters $\{C_1, \ldots, C_k\}$ and a flow $Flow_j = (t_j, G_j)$ (with $j = 1, \ldots, g$).

- $t_j \in hour_i$ if the $Flow_j$ starts in the hour i;
- $Flow_j \in C_i$ if the $Flow_j$ belongs to cluster C_i;
- $D(G_j, C_i)$ the euclidean distance between the vector of the flow features G_j representing $Flow_j$ and the mean of the cluster C_i.

The hourly-based features consist of a set of features corresponding to the different clusters, for each hour $hour_i$ of host h_i (Figure 4.2). First, given an hour $hour_i$, MINOS extracts three features for each cluster C_l defined as follows:

1. The number of flows of host h_i starting in $hour_i$ and belonging to cluster C_l, i.e. $|\{Flow_j | j \in \{1, \ldots, g\}, t_j \in hour_i, Flow_j \in C_l\}|$.
2. The summation of the Euclidean distances of all the flows of host h_i starting in $hour_i$ and belonging to cluster C_l, i.e. $\sum_{j\in\{1,\ldots,g\}, t_j \in hour_i, Flow_j \in C_l} D(G_j, C_l)$.
3. The maximum Euclidean distance among all the flows of host h_i starting in $hour_i$ and belonging to cluster C_l, i.e. $\max_{j\in\{1,\ldots,g\}, t_j \in hour_i, Flow_j \in C_l} D(G_j, C_l)$.

The last two features capture how anomalous are the traffic flows belonging to cluster C_l in the hour $hour_i$ according to two different aggregations (i.e., θ_{max} and θ_{avg} see section 4.4). Besides, we define two other features as follows:

1. The summation for each flows $Flow_j$ starting in $hour_i$ of the minimum euclidean distances $D(G_j, C_l)$ for each cluster C_l, i.e. $\sum j \in \{1,...,g\}, t_j \in hour_i \min l \in \{1,...,k\} D^{(}Gj, Cl^{)}$
2. The maximum for each flows $Flow_j$ of host h_i starting in $hour_i$ of the minimum Euclidean distances $D(G_j, C_l)$ for each cluster C_l, i.e. $\max j \in \{1,...,g\}, t_j \in hour_i \min l \in \{1,...,k\} D^{(}Gj, Cl^{)}$

We introduce these two additional features as anomaly indicators that are computed while considering all the clusters together. In summary, the total number of hourly-based features (Figure 4.2) is $3 * k + 2$ for each hour $hour_i$ of host h_i. There are 3 features per k cluster. Also, two additional features exist, considering all the clusters together. As the last step, once the hourly feature matrix is obtained, we perform standardization of each feature, so that the elements have a mean of 0 and a standard deviation of 1.

Feature Extraction Algorithm. We describe our feature extraction procedure in the Algorithm 1 that takes two inputs to produce the list for feature vector ($hour_d^N$). Hence, N represents the selected machines (1,000 in number). The input parameters are $Flow_j^N$ and d. The $Flow_j^N$ represents the input flows matrix of all machines, and d represents the total hours in the given time frame. Since there are a total 8, 784 hours in the selected year the value of d should be 8, 784.

Algorithm 1 Feature Vector Extraction

```
1: Input
2:     $Flow_j^N$ Flows matrices of all machines
3:     $d$ The calculative hours into given time period
4: Output
5:     $hour_d^N$ Feature vectors for all machines
6: procedure FVE($Flow_j^N$, $d$)
7:     $hour_d^N \leftarrow \emptyset$
8:     for $i \leftarrow 1$ to $N$ do
9:         $k \leftarrow \left\lceil \sqrt{\frac{Flow_j^i}{2}} \right\rceil$    ▷ The number of clusters
10:        $mlModel \leftarrow ModelFitting(k, Flow^i_j)$
11:        $hour_d^i \leftarrow ScoreTransform(Flow^i_j, mlModel)$
12:        $hour_d^N \leftarrow hour_d^N \cup hour_d^i$
13:    end for
14: return $hour_d^N$.    ▷The list of feature matrix for all machines
15: end procedure
16: function MODELFITTING($k$, $Flow^i_j$)
17:     $mlModel \leftarrow KMeansFitting(k, Flow^i_j)$
18:     return $mlModel$    ▷The fitted Kmeans model
19: end function
20: function SCORETRANSFORM($Flow^i_j$, $mlModel$)
21:     for $p \leftarrow 1$ to $d$ do
22:         $l_m, s_m \leftarrow \emptyset$
23:         $l_b \leftarrow mlModel.Predict(Flow^i_p)$
24:         $l_b \leftarrow processed(\overline{l_b})$
25:         $\overline{t_m} \leftarrow mlModel.Transform(Flow^i_p)$
26:         $t_m \leftarrow processed(\overline{t_m})$
27:         $sf_1 \leftarrow [max(\exists k \forall Flow^i_p \in t_m)]$
28:         $sf_2 \leftarrow [\sum(\exists k \forall Flow^i_p \in t_m)]$
29:         $ls \leftarrow [max((\exists Flow^i_p \forall k) \in t_m)]$
30:         $sf_3 \leftarrow max(ls)$
31:         $sf_4 \leftarrow \sum(ls)$
32:         $s_m \leftarrow l_b \cup sf_1 \cup sf_2 \cup sf_3 \cup sf_4$
33:         $l_m \leftarrow l_m \cup s_m$
34:     end for
35:     return $l_m$    ▷ The feature matrix of a single machine
36: end function
```

The feature extraction procedure has the following two steps. Firstly, train a machine learning model for input dataset. Secondly, prepare a meaningful feature matrix after transforming the score of that model so it can be a high learning point for the anomaly detection algorithm in the future.

For the first step, the algorithm called "*ModelFitting*" function, which is utilized to fit the $K - means$ model using the k and $Flow^i_j$ parameters. The returns from the mentioned function declare the object of a trained $k - means$ model, "*mlModel*". While the model becomes ready, the next target is to construct a feature vector from the created model.

The *ScoreTransform* function starts with a *for* loop running from 1 to d, where each iteration represents an hour, a row of the feature matrix for the i^{th} machine. Afterward, the trained model (*mlModel*) predicts the number of clusters. The following matrix draws a sample structure of the predicted matrix $(\overline{l_b})$ with a dimension of $1 \times G$, where each column represents the number of a cluster for a unique flow.

$$\overline{l_b} = \begin{bmatrix} C_k & C_1 & C_1 & C_3 & C_2 & C_3 & C_2 & C_3 \dots C_5 \end{bmatrix}$$

Our objective is to extract some meaningful features such that the novelty detection algorithm can learn as much as possible to produce the desired score. Thus, we modified the predicted matrix depending on the total number of flows into each cluster. Therefore, the matrix can identify a normal (highest number of flows belonging to a cluster) and an abnormal (lowest number of flows belonging to a cluster) cluster. Generally, the distinguishing between normal and abnormal flows cluster can be a potential learning foundation for the anomaly detection algorithm. The following vector l_b with a dimension of $1 \times k$, represents our processed matrix of $\overline{l_b}$.

$$l_b = \begin{bmatrix} 2 & 2 & 3 & 0 & 1 & 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Since there is a total of k clusters, we can represent the feature vector of l_b in the following way:

$$lb = \begin{bmatrix} f_p^1 & f_p^2 & f_p^3 & \cdots & f_p^k \end{bmatrix}$$

In the next step, our algorithm generates a transform matrix (t_m) by transforming the flows of the current hour using the trained *mlModel*. The transform property used to return a cluster distance such that in the new space, each dimension represents the distance to the cluster centers. Therefore, after the transformation of each flow, we obtain a distance-vector with the cardinality $|C_k|$. The following matrix (t_m) illustrates the outlook of the transform matrix, where D denotes the distance to the cluster center. For example, $D(2, 3)$ describes itself as the distance of the second flow from the center of the third cluster.

$$\overline{\mathbf{t_m}} = \begin{bmatrix} D(1,1) & D(1,2) & D(1,3) & \cdots & D(1,k) \\ D(2,1) & D(2,2) & D(2,3) & \cdots & D(2,k) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D(g,1) & D(g,2) & D(g,3) & \cdots & D(g,k) \end{bmatrix}$$

We processed the above transform matrix t_m such that it keeps a record of the minimum distance of each flow rather than the whole distance. Logically, if cluster C_k contains flow 1, then the minimum distance from flow 1 to any cluster's centroid would be the distance from C_k to flow 1. In this case, we define zero for other clusters except the C_k, which constructs the following matrix t_m:

$$\left[\begin{array}{ccccc} C_1 & C_2 & C_3 & \dots & C_k \\ 0 & 0 & 0 & \dots & D(1,2) \\ D(2,1) & 0 & 0 & \dots & 0 \\ D(3,1) & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{array}\right]$$

Afterward, we compute a maximum and summation of all distances per cluster in the t_m matrix to define the depth of the anomaly.

$f_p^{k+1} = \max(0, D(2, 1), D(3, 1), \dots, 0)$ $f_p^{2k+1} = sum(0, D(2, 1), D(3, 1), \dots, 0)$

$f_p^{k+2} = \max(0, 0, 0, \dots, 0)$ $f_p^{2k+2} = sum(0, 0, 0, \dots, 0)$

$\vdots$

$f_p^{k+k} = \max(D(1, g), 0, 0, \dots, 0)$ $f_p^{2k+k} = sum(D(1, g), 0, 0, \dots, 0)$

These maximum and summation values are the primary materials for producing sub-feature vector sf_1 and sf_2 with a dimension of $1 \times k$, mentioned in line number 27 and 28 in the Algorithm 1.

$$sf_1 = \begin{bmatrix} f_p^{k+1} & f_p^{k+2} & \dots & f_p^{k+k} \end{bmatrix}$$

$$sf_2 = \begin{bmatrix} f_p^{2k+1} & f_p^{2k+2} & \dots & f_p^{2k+k} \end{bmatrix}$$

Our next step is to create a list of minimum distances per flow for all clusters from the matrix t_m. The following equations are the mathematical representation of this task, where $minF_1$ expresses the minimum distance from the first flow to all clusters in t_m matrix.

$$minF_1 = \max(0, 0, \ldots, D(1, k))$$

$$minF_2 = \max(D(2, 1), 0, \ldots, 0)$$

$$\vdots$$

$$minF_g = \max(0, 0, \ldots, 0)$$

Line number 29 in Algorithm 1 is exploring itself as a sequence of those minimum values which we computed above ($minF_1, \ldots, minF_g$).

$$ls = [minF_1, minF_2, \ldots, minF_g]$$

We create two sub-features (sf_1 and sf_2) out of four. The next target is to build the rest of the sub-features (sf_3, sf_4) to create a feature matrix for a specific window. The complexity of sf_3 and sf_4 are not as similar as sf_1 and sf_2.They are just the calculation of maximum and summation of the list (ls), which the algorithm produced in line number 30 and 31. The following two equations are the formal presentation of these sub-features:

$$sf_3 = \max(ls)$$

$$sf_4 = \sum_{i=1}^{n}(ls[i])$$

We use the prediction label matrix (lb) and all sub-features (sf_1, sf_2, sf_3, and sf_4) to form a sample (s_m) of feature vector. The dimension of this sample is (s_m) is $1 \times m$. Algorithm 1 produces d number of samples (s_m). Line number 33 in Algorithm 1 horizontally merges each of the samples to build the large matrix (l_m). At the end, the function *ScoreTransform* returns l_m as the final feature vector. Hence, the dimension of this final feature vector would be $d \times m$. The following matrix illustrates the final feature vector of a single machine:

$$\begin{bmatrix} f_1^1 \dots & f_1^k & f_1^{k+1} \dots & f_1^{k+k} & f_1^{2k+1} \dots & f_1^{2k+k} & f_1^{3k+1} & f_1^{3k+2} \\ f_2^1 \dots & f_2^k & f_2^{k+1} \dots & f_2^{k+k} & f_2^{2k+1} \dots & f_2^{2k+k} & f_2^{3k+1} & f_2^{3k+2} \\ \vdots \ \ddots & \vdots & \vdots \ \ddots & \vdots & \vdots \ \ddots & \vdots & \vdots & \vdots \\ f_d^1 \dots & f_d^k & f_d^{k+1} \dots & f_d^{k+k} & f_d^{2k+1} \dots & f_d^{2k+k} & f_d^{3k+1} & f_d^{3k+2} \end{bmatrix}$$

However, the above matrix stands only for a single machine. As we are considering 1000 machines, the algorithm generates a list of 1000 feature vectors. Line number 13 of the algorithm constructs this final matrix by aggregating all the single machines feature vector.

In summary, we described in this section how we created a well-defined feature vector. In the next part, we will articulate the application procedure of the novelty detection algorithm over this feature vector.

4.2 Step 3: Hourly-based Anomaly Detection Model

We use anomaly detection techniques to identify non-conforming hours inside the time frame of a host. An anomaly detection technique learns the normal hourly behavior of the host and gives to each hour an anomaly detection score. The higher the anomaly score for $hour_i$, the higher the likelihood that $hour_i$ is anomalous. MINOS creates an anomaly detection model, and the output of that model is a vector of anomaly scores, one anomaly score for each hour of the time frame. For the anomaly detection model, we considered the Gaussian Mixture Model (GMM), One-Class Support Vector Machine (OC-SVM), Autoencoder, and Long Short-Term Memory (LSTM) Autoencoder.

Gaussian Mixture Model is an advanced clustering technique that works by learning a mixture of multivariate Gaussian distributions where each distribution represents a specific cluster. Once trained, it assigns to each point a probability that the mixture distribution generates the points. The anomaly score for this technique is obtained, not by the learned Gaussian Mixture distribution, but by computing this probability.

One-Class Support Vector Machine [38] is a classification algorithm based on the binary support vector machine with the peculiarity to use only one class in the training phase. The binary classification is obtained by analyzing the sign of the decision function: if positive, the class is the same as the example used in training, and if negative, the class is different. The anomaly score is obtained by inverting the sign of the decision function.

Autoencoder is very similar to a feed-forward multilayer perceptron neural network. The encoder part aims to learn an encoded representation (embeddings) of training in different feature space data by efficiently reducing the dimensionality of the original data space. The decoder phase tries to reconstruct the original data by taking the embeddings (compressed feature vectors) as input. For our feature vector, we encoded and decoded twice. The output of an autoencoder has the same number of computational units as the input (original feature dimension). An autoencoder reconstructs hourly-based features that have similar statistical properties in the original feature space. Smaller reconstruction errors represent the normal and higher reconstruction errors represents the anomalous hourly-based features.

LSTM Autoencoder [4] is an autoencoder where the input and output are the sequences of hourly-based features. We considered input sequences of consecutive 24 hours and created a dataset of sequences obtained by shifting each sequence by one hour.

$$\left|\begin{array}{c} < hour_1 = [f_1^1, \dots, f_1^m], \dots, hour_{24} = [f_{24}^1, \dots, f_{24}^m] > \\ < hour_2 = [f_2^1, \dots, f_2^m], \dots, hour_{25} = [f_{25}^1, \dots, f_{25}^m] > \\ \\ < hour_{d-24} = [f_{d-24}^1, \dots, f_d^m{}_{-24}], \dots, hour_d = [f_d^1, \dots, f_d^m] > \end{array}\right|$$

The LSTM Autoencoder reconstructs each sequence and assigns to each sequence a reconstruction error. The dataset has $d - 24$ sequences, which produces $d - 24$ anomaly scores.

4.3 Step 4: Anomaly Scores Normalization and Aggregation

The LSTM Autoencoder reconstructs each sequence and assigns to each sequence a reconstruction error. The dataset has $d - 24$ sequences, which produces $d - 24$ anomaly scores.

4.4 Step 4: Anomaly Scores Normalization and Aggregation

To obtain a single score for each host, we use the following two aggregation procedures:

- *Max Aggregation* (θ_{max}): Standardizing the score vector (with mean 0 and standard deviation 1), and then computing the maximum of the normalized anomaly scores among all the hours of host h_i.
- *Avg Aggregation* (θ_{avg}): Scaling the score vector by dividing each component by the maximum absolute value, and then computing the average of all the scaled anomaly scores among all the hours of host h_i.

The θ_{max} and θ_{avg} represents the worst and average anomalous scenario, respectively, for each host. We compute the scores of each host individually and compare them with the normalization operations (the standardization and the scaling operation).

The maximum absolute scaling and standardization (i.e., the two normalization procedures) are not interchangeable in the θ_{max} and θ_{avg}, because the average of different standardized data would always correspond to the same value and the maximum absolute value of different scaled data may always correspond to value 1.

4.5 Step 5: Unsupervised Classification

The detection module group obtained anomaly scores for all hosts by using K-Means (number of clusters k=3). It assigns all the hosts inside a cluster to a class among CH, HUA, and MH. The centroid with the lowest value is assigned to the class of clean hosts (CH), the one with the highest value is assigned to the class of malicious hosts (MH), and the centroid with the middle value is assigned to the class of hosts under attack (HUA).

4.6 Step 6: Anomalous or Not Anomalous Identification

After standardizing the score vector of all hosts, the detection module identifies each hour of the host as either Anomalous or Not Anomalous. If the score of a particular hour is higher (positive), that hour is classified as the anomalous time frame. On the other hand, if the score of a particular hour is lower (negative), that hour is identified as the not anomalous time frame.

CHAPTER FIVE: EXPERIMENTAL EVALUATION

In this chapter, we evaluate the capability of MINOS to prioritize hosts among CH, HUA, and MH based on anomaly scores. Afterward, we measure the accuracy and required execution times at different time frames followed by unsupervised classifications of each host as either CH, HUA, and MH. Finally, we identify precisely when any particular attack or list of the attacks has happened for HUA and MH hosts.

We also compare our research outcomes concerning MINOS with Kitsune's [41], state of the art approach. As discussed in Section 2.7, Kitsune was originally designed to analyze packets data of IoT traffic for a few hours. However, there are two main reasons this method is not valid in large networks. First, it is infeasible to analyze the packet level information for a given period (i.e., a couple of months / a year/ more). Second, it is insufficient to create a behavioral model for complicated activities and large patterns [2, 39] of client/server hosts. Hence, we reimplement the same architecture and methodology of Kitsune (based on an ensemble of Autoencoders), but we apply it on our hourlybased features (cf. Section 4.1).

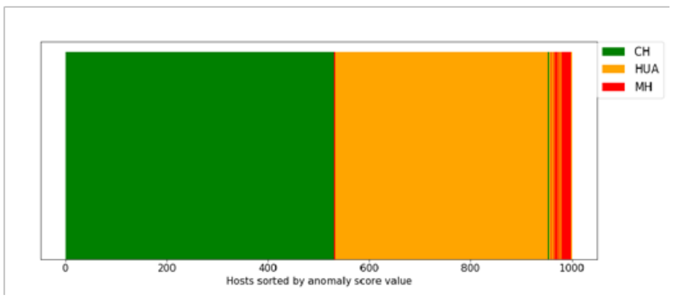

Figure 5.1 Hosts sorted by increasing OC-SVM anomaly score value.

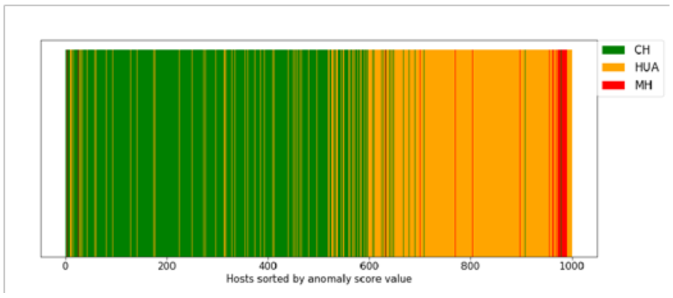

Figure 5.2 Hosts sorted by increasing Autoencoder anomaly score value.

5.1 Separating CH, HUA, and MH with Hourly-based Anomaly Scores

We first evaluate the capability of the anomaly scores to separate the hosts within the three categories (CH, HUA, and MH). Figures 5.1, 5.2, 5.3 report three bar charts (for OC-SVM, Autoencoder and Kitsune) with the θ_{avg} anomaly scores on the time frame of one year. The hosts are ordered over the X-axis according to the anomaly scores produced by MINOS. Each vertical bar represents a specific host, and the color represents the ground truth of one of three classes among CH, HUA, and MH. Figure 5.1 shows that the anomaly score mostly orders first the clean hosts (CH), second the hosts under attack (HUA) and last the malicious hosts (MH). In other words, malicious hosts have the

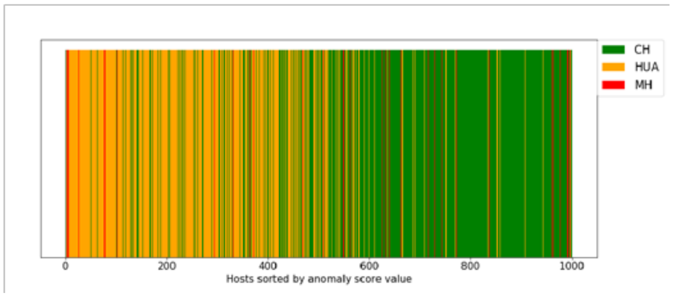

Figure 5.3 Hosts sorted by increasing Kitsune anomaly score value.

highest anomaly scores. This figure intuitively explains how MINOS can effectively prioritize the riskiest and most dangerous hosts in the network. We can observe that OC-SVM offers the best separation of the three classes.

To better quantify the discriminatory powers of the MINOS anomaly scores we consider two binary classification problems: (i) *CH vs. Other* (i.e., CH vs. HUA and MH), and then (ii) *HUA vs. MH*. In particular, in the case of *CH vs. Other*, we test the hypothesis that higher anomaly scores correspond to a higher likelihood that the host is under attack or malicious. In the case of *HUA vs. MH,* we test the hypothesis that higher anomaly scores correspond to the higher likelihood of a host being malicious. To test these hypotheses, we use the *Area Under the Receiver Operating Characteristic* curve (AUROC) and the *Average Precision* (AP).

Table 5.1 reports the results for the different anomaly scores obtained over a period of 12 months with the different anomaly models and two different aggregation procedures (θ_{max} and θ_{avg}). Table 5.1 shows that the OC-SVM with the θ_{avg} aggregation in the MINOS anomaly detection approach satisfies all the hypotheses. After the OC-SVM, the Autoencoder works perfectly for the case CH vs. Other, but not for HUA vs. MH. MINOS (OC-SVM) also outperforms Kitsune, especially in terms of AP, and in the case of HUA vs. MH (Figures 5.1, 5.2 and 5.3).

Table 5.1 AUROC and AP of anomaly scores on 12 months of traffic. Values ≥ 0.80 are highlighted in bold.

	CH vs. Other				HUA vs. MH			
	AUROC		AP		AUROC		AP	
Algorithm/Aggregation	θ_{max}	θ_{avg}	θ_{max}	θ_{avg}	θ_{max}	θ_{avg}	θ_{max}	θ_{avg}
MINOS (GM M)	**0.95**	0.05	**0.97**	0.31	0.04	0.39	0.07	0.18
MINOS (OC-SVM)	0.20	**1.00**	0.37	**1.00**	**0.81**	**0.90**	0.55	0.71
MINOS (Autoencoder)	**1.00**	**0.92**	**0.99**	**0.93**	0.30	0.72	0.12	0.29
MINOS (LSTM)	**0.82**	0.18	0.75	0.32	**0.85**	0.13	0.38	0.04
Kitsune [41]	**0.99**	0.12	**0.99**	0.31	0.29	0.69	0.13	0.30

5.2 Attack Time Identification

We described the ground truth collection per hour for each machine in chapter 4. In this section, we explain our results of recognizing when an attack has happened for HUA and MH by graphical representation. For attack time identification we use both HUA and MH to compare hourly based experimental anomaly detection score with the ground truth and then produce the accuracy score of AUROC and AP.

Figures 5.4, 5.5, 5.6, and 5.7 represent the accuracy of AUROC and AP against the experimental result of $LSTM - Autoencoder$ and $OC - SVM$ for HUA and MH, respectively. The horizontal line indicates all machines, and the vertical line implies the accuracy score. The red and blue line indicates the accuracy of AP and AUROC, respectively.

Table 5.3 shows that, among all the experiments, the best accuracy for host

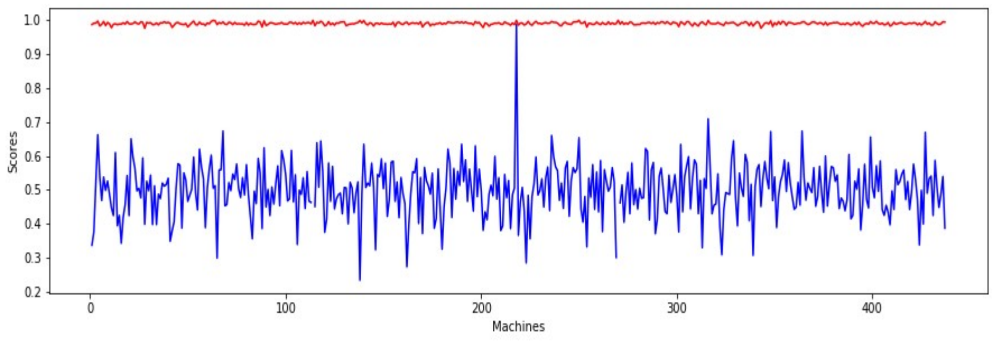

Figure 5.4 The LSTM accuracy of attack time for the HUA

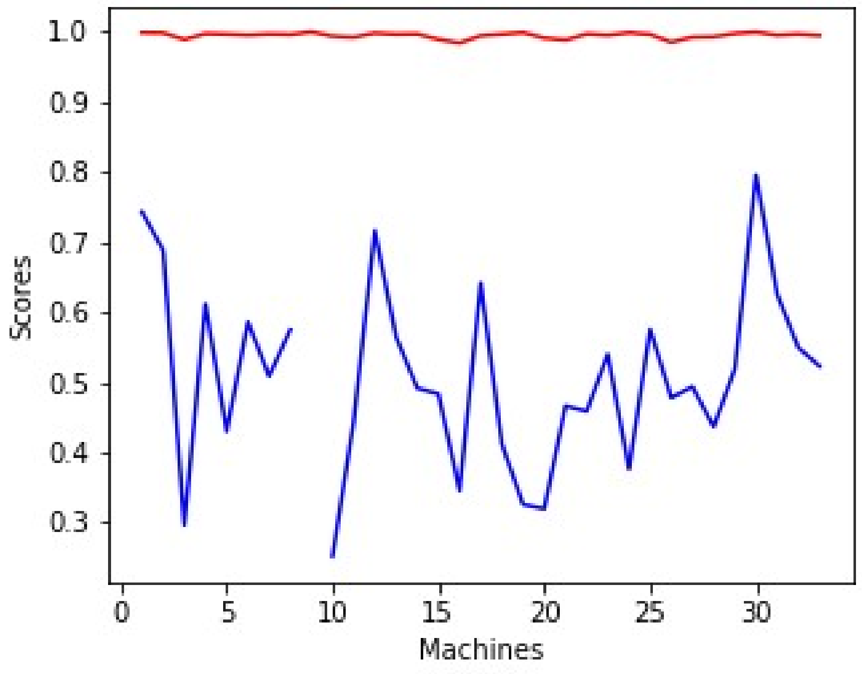

Figure 5.5 The LSTM accuracy of attack time for the MM

identification was achieved by the $OC - SVM$. For the attack time identification, we found that, among all of them, LSTM-Autoencoder and OC-SVM performed the best. Figure 5.4 represents the accuracy for HUA, where the accuracy of AP reached up to 100%, and the efficiency of AUROC fluctuated

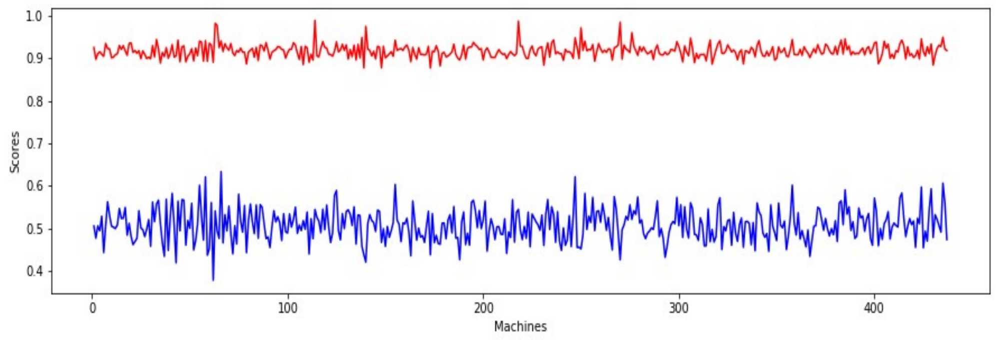

Figure 5.6 The OC-SVM accuracy of attack time for the HUA

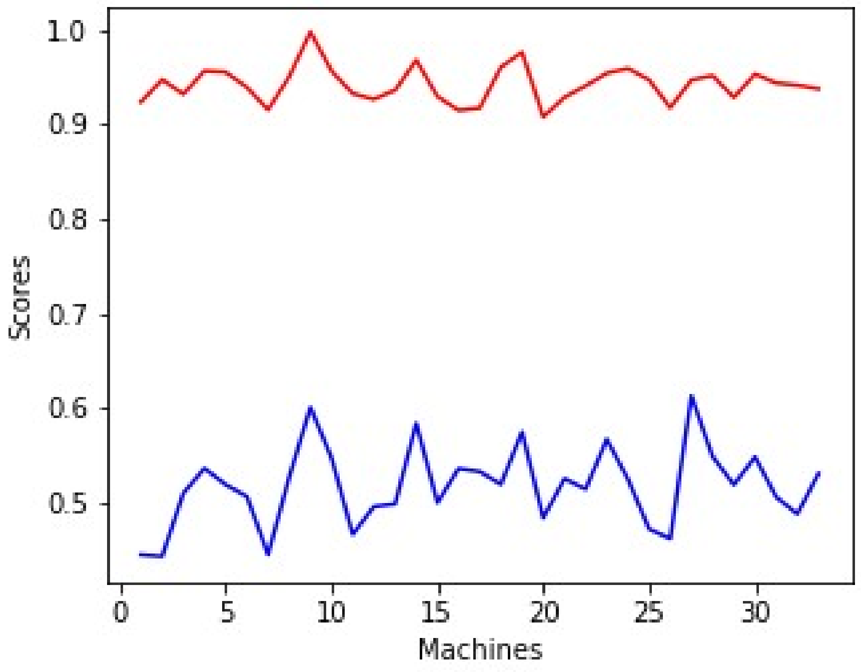

Figure 5.7 The OC-SVM accuracy of attack time for the MH

between 60% to 80%. In some cases, even AUROC fell sharply; because some machines are carrying a few attacks in most of the sequences. Since our LSTM

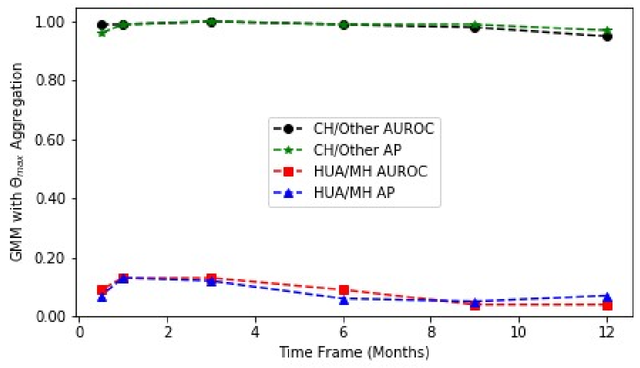

Figure 5.8 AUROC and AP of GMM for different time frames.

anomaly detection algorithm used 24 hour sequences to train the model, we aggregated 24 hour ground truth each time for the $LSTM - Autoencoder$. Therefore, for some machines, there exist a few attacks for most of the sequences, which creates minimal accuracy for the AUROC. On the other hand, figure 5.5 depicts that there is not available AUROC accuracy for a few machines, because those machines contained at least one attack for all of the sequences. Thus, AUROC produced "nan" value instead of any accuracy.

5.3 Different Time Frames

We investigate the impact of time frame size on the accuracy of the MINOS anomaly scores. Figures 5.8, 5.9, 5.10, and 5.11 report MINOS scores for all the anomaly detection models (y-axis), corresponding to 0.5, 1, 3, 6, 9, and 12 months of time frame (x-axis). These plots show that OC-SVM with the θ_{avg} aggregation is the best anomaly detection model, even when the analysis dataset size is only one month, in which AUROC is higher than 0.8 for both separation problems (i.e., CH vs. Other and HUA vs. MH).

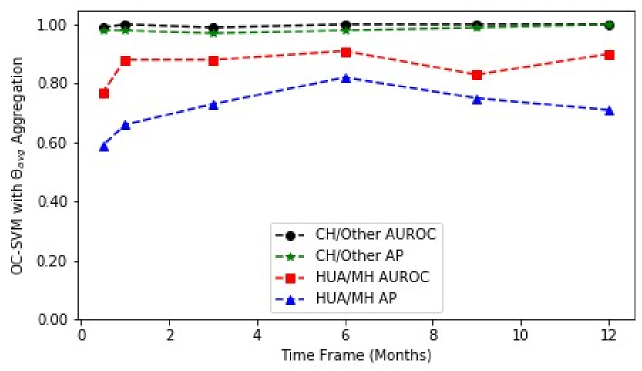

Figure 5.9 AUROC and AP of OC-SVM for different time frames.

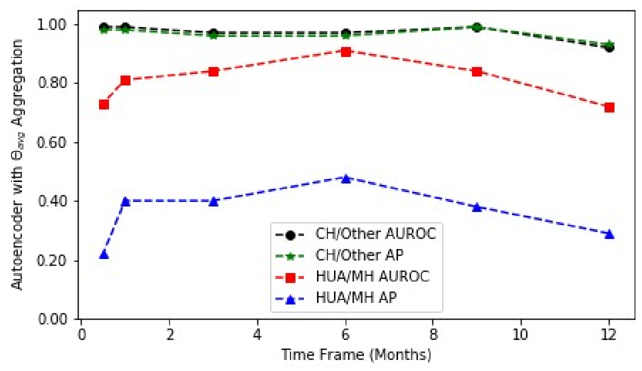

Figure 5.10 AUROC and AP of Autoencoder for different time frames.

5.4 Execution Time

Since the dataset is massive, Titan has been used to run the proposed approaches. Titan is a supercomputer that uses Graphics Processing Units (GPUs) including conventional Central Processing Units (CPUs). Titan's performance is measured in floating-point operations per second (FLOPS) instead of million instructions per second (MIPS). Titan is the first such hybrid to perform over ten petaFLOPS.

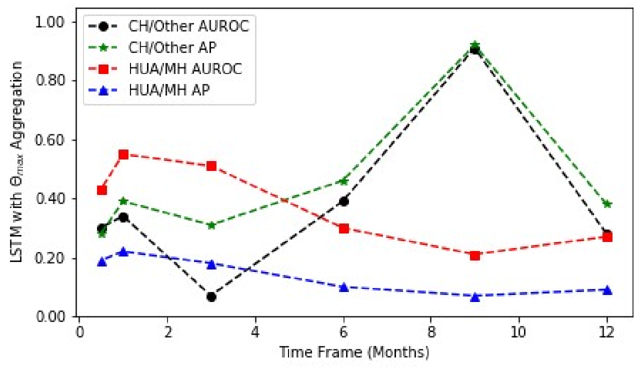

Figure 5.11 AUROC and AP of LSTM for different time frames.

Table 5.2 Execution time (in minutes) for 1,000 machines.

	Time Frame Analyzed					
Algorithm	2 Weeks	1 Month	3 Months	6 Months	9 Months	12 Months
MINOS (GMM)	2.59	6.70	213.90	410.35	945.49	2568.88
MINOS (OC-SVM)	**1.58**	**3.71**	**107.02**	**245.76**	**402.62**	**597.63**
MINOS (Autoencoder)	66.88	77.82	226.86	627.48	943.98	1663.88
MINOS (LSTM)	178.56	392.14	1208.99	2949.78	4386.78	6498.38
Kitsune [41]	69.12	80.86	234.78	652.14	970.87	1728.14

In our experiments, we used a server with two 2.10 GHz Xeon E4-2620 Processors, 128GB RAM, and 4 Titan X GPUs. The system ran Ubuntu 16.4 with scikit-learn, Keras and ThunderSVM libraries. In particular, Keras and ThunderSVM libraries can use Titan X GPU to speed up the computational time (inference and test) for GMM, Autoencoder, LSTM, and One-Class Support Vector Machine (OC-SVM).

Table 5.2 reports the total execution times (in minutes) to analyze 1,000 hosts with MINOS and Kitsune, where MINOS (OC-SVM) is fastest, followed by Autoencoder. Also, Kitsune has higher execution time with lower detection performance.

Table 5.3 Performance of 3-class detection module (12 months).

		θmax			*θavg*		
Algorithm	Metric	Prec.	Rec.	F_1	Prec.	Rec.	F_1
MINOS (GMM)	Micro	0.01	0.01	0.01	0.44	0.44	0.44
	Macro	0.09	0.04	0.01	0.33	0.35	0.24
	Weighted	0.14	0.01	0.01	0.32	0.44	0.28
MINOS (OC-SVM)	Micro	0.54	0.54	0.54	0.98	0.98	0.98
	Macro	0.54	0.36	0.29	0.92	0.90	0.91
	Weighted	0.48	0.54	0.40	0.98	0.98	0.98
MINOS (Autoencoder)	Micro	0.88	0.88	0.88	0.81	0.81	0.81
	Macro	0.66	0.65	0.64	0.65	0.58	0.60
	Weighted	0.92	0.88	0.90	0.82	0.81	0.79
MINOS (LSTM)	Micro	0.69	0.69	0.69	0.30	0.30	0.30
	Macro	0.57	0.61	0.57	0.26	0.47	0.26
	Weighted	0.70	0.69	0.68	0.35	0.30	0.31
Kitsune [41]	Micro	0.09	0.09	0.09	0.16	0.16	0.16
	Macro	0.32	0.16	0.11	0.14	0.18	0.14
	Weighted	0.41	0.09	0.13	0.18	0.16	0.17

5.5 Unsupervised Classification Performance

Table 5.3 reports the performance results (Precision, Recall, F_1-Score) when using the fully unsupervised classification module of MINOS described in Section 4.5. Since there are three classes (CH, HUA, MH), we report Micro, Macro, and Weighted statistics. These results confirm that MINOS with OC-SVM and θ_{avg} aggregation achieves the best performance. We remark that no training labels have been used by MINOS to achieve this performance. The lower Macro performance in OC-SVM is related to some malicious hosts classified as under attack and vice versa, while the separation between CH and the others (HUA and MH) remains very nitid. Conversely, Kitsune [41] performs poorly because there is not a clear separation between the anomaly scores it generates (cf. Figure 5.1,5.2, and 5.3).

CHAPTER SIX: CONCLUSION

6.1 Summary

We have proposed MINOS, a *fully unsupervised* method for traffic anomaly detection, which does not require any ground truth or label data. It can perform offline analysis for large networks efficiently. Moreover, it can prioritize and classify internal hosts in three categories: clean hosts, hosts under attack, and malicious hosts. Also, it can identify the time frame of an attack for malicious and under attack machines. MINOS with OC-SVM and θ_{avg} aggregation method performs better than state of the art, both in terms of accuracy and execution time.

We remark that MINOS can parallelize each host separately and analyze large time frames of traffic in a short time. The low execution times suggest that future work can effectively adapt MINOS for online analysis.

The proposed methodology is obtaining high accuracy by analyzing normal traffic, where the status of a machine is entirely unknown. In this context, *Chapter 6. Conclusion*

the low false alarms (false positive rates) of this unsupervised novelty detection score can be a practical solution to the problem, which is consistent with safety and security. Also, our work proves that only two weeks of traffic flows are sufficient to obtain the desired result in a year. As a result, it reduces the execution time and acts as a less resource-intensive task.

6.2 Future Work

We used Gaussian Mixture Model, One-Class Support Vector Machine, Autoencoder, and LSTM-Autoencoder methodologies for our Anomaly detection models. We believe that there is a space for more improvement in terms of differentiating malicious machines and machines under attack. Moreover, an hourly-based feature extraction procedure needs two days for 1000 machines to create a feature vector. We can improve the execution time of this procedure. Furthermore, instead of only offline analysis, we can think about online analysis. We need to enhance the accuracy of the attack time identification for a machine. Additionally, we are identifying the host and time of the attack in an unsupervised manner in this research, which could open up the possibility of also defining a type of priority of the attack in an unsupervised way.

BIBLIOGRAPHY

[1] Hashem Alaidaros, Massudi Mahmuddin, Ali Al-Mazari, et al. “An overview of flow-based and packet-based intrusion detection performance in high speed networks”. In: (2011).

[2] Giovanni Apruzzese et al. “Detection and threat prioritization of pivoting attacks in large networks”. In: *IEEE Transactions on Emerging Topics in Computing* (2017).

[3] Stefan Axelsson. “The base-rate fallacy and the difficulty of intrusion detection”. In: *ACM Transactions on Information and System Security (TISSEC)* (2000).

[4] Inci M Baytas et al. “Patient subtyping via time-aware LSTM networks”. In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2017, pp. 65–74.

[5] Elisa Bertino and Gabriel Ghinita. “Towards mechanisms for detection and prevention of data exfiltration by insiders: keynote talk paper”. In: *ACM ICCS*. 2011.

[6] Richard J Bolton, David J Hand, et al. “Unsupervised profiling methods for fraud detection”. In: *Credit Scoring and Credit Control VII* (2001), pp. 235–255.

[7] Anna L Buczak and Erhan Guven. “A survey of data mining and machine learning methods for cyber security intrusion detection”. In:

IEEE Communications Surveys & Tutorials 18.2 (2016), pp. 1153–1176.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: A Survey”. In: *ACM Comput. Surv.* 41.3 (July 2009), 15:1–15:58. ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: http://doi.acm.org/10.1145/1541880.1541882.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), p. 15.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Outlier detection: A survey”. In: *ACM Computing Surveys* (2007).

[11] Benoit Claise, Brian Trammell, and Paul Aitken. *Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information*. Tech. rep. 2013.

[12] Gilles Cohen, Melanie Hilario, and Christian Pellegrini. “One-class support vector machines with a conformal kernel. a case study in handling class imbalance”. In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR).* Springer. 2004, pp. 850–858.

[13] Xuemei Ding et al. “An experimental evaluation of novelty detection methods”. In: *Neurocomputing* 135 (2014), pp. 313–327.

[14] Holger Dreger et al. “Operational experiences with high-volume network intrusion detection”. In: *Proceedings of the 11th ACM*

conference on Computer and communications security. ACM. 2004, pp. 2–11.

[15] Computer Economics. “Malware report: The economic impact of viruses, spyware, adware, botnets, and other malicious code”. In: *Computer Economics* (2007).

[16] Zakia Ferdousi and Akira Maeda. “Anomaly Detection Using Unsupervised Profiling Method in Time Series Data.” In: *ADBIS Research Communications*. 2006.

[17] Ming Gao, Kenong Zhang, and Jiahua Lu. “Efficient packet matching for gigabit network intrusion detection using TCAMs”. In: *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*. Vol. 1. IEEE. 2006, 6–pp.

[18] Pedro Garcia-Teodoro et al. “Anomaly-based network intrusion detection: Techniques, systems and challenges”. In: *computers & security* 28.1-2 (2009), pp. 18–28.

[19] Markus Goldstein and Seiichi Uchida. “A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data”. In: *PloS one* 11.4 (2016), e0152173.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[21] Guofei Gu et al. “Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection”. In: (2008).

[22] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[23] Rasha G Mohammed Helali. “Data mining based network intrusion detection system: A survey”. In: *Novel Algorithms and Techniques in Telecommunications and Networking*. Springer, 2010, pp. 501–505.

[24] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504– 507.

[25] Rick Hofstede et al. “Flow monitoring explained: From packet capture to data analysis with netflow and ipfix”. In: *IEEE Communications Surveys & Tutorials* 16.4 (2014), pp. 2037–2064.

[26] Natthakan Iam-On and Tossapon Boongoen. “Comparative study of matrix refinement approaches for ensemble clustering”. In: *Machine Learning* 98.1 (Jan. 2015), pp. 269–300. ISSN: 1573-0565. DOI: 10.1007/s10994-013-5342-y.

[27] Anil K Jain. “Data clustering: 50 years beyond K-means”. In: *Pattern recognition letters* 31.8 (2010), pp. 651–666.

[28] Ahmad Javaid et al. “A deep learning approach for network intrusion detection system”. In: *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ICST (Institute for Computer Sciences, Social-Informatics and ... 2016, pp. 21–26.

[29] Harjinder Kaur, Gurpreet Singh, and Jaspreet Minhas. “A review of machine learning based anomaly detection techniques”. In: *arXiv preprint arXiv:1307.7286* (2013).

[30] KDD. *Dataset*. kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. 1999.

[31] Richard A Kemmerer and Giovanni Vigna. “Intrusion detection: a brief history and overview”. In: *Computer* 35.4 (2002), supl27–supl30.

[32] Marshall A Kuypers, Thomas Maillart, and Elisabeth Pate-Cornell. “An empirical analysis of cyber security incidents at a large organization”. In: *Department of Management Science and Engineering, Stanford University, School of Information, UC Berkeley, http://fsi.stanford.edu/sites/default/files/kuypersweis_v7.pdf, accessed July* 30 (2016).

[33] MIT Lincoln Lab. *Darpa Datasets*. https://www.ll.mit.edu/r-d/datasets. 1998–2000.

[34] Haiguang Lai et al. “A parallel intrusion detection system for high-speed networks”. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2004, pp. 439–451.

[35] Han Li. “Research and implementation of an anomaly detection model based on clustering analysis”. In: *Intelligence Information Processing and Trusted Computing (IPTC), 2010 International Symposium on*. IEEE. 2010, pp. 458–462.

[36] Yali Liu et al. “SIDD: A framework for detecting sensitive data exfiltration by an insider attack”. In: *Hawaii International Conference on System Sciences*. IEEE. 2009.

[37] Matthew V Mahoney and Philip K Chan. “An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection”. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2003, pp. 220–237.

[38] Larry M Manevitz and Malik Yousef. “One-class SVMs for document classification”. In: *Journal of machine Learning research* 2.Dec (2001), pp. 139– 154.

[39] Mirco Marchetti et al. “Analysis of high volumes of network traffic for advanced persistent threat detection”. In: *Computer Networks* (2016).

[40] Markos Markou and Sameer Singh. “Novelty detection: a review—part 1: statistical approaches”. In: *Signal processing* 83.12 (2003), pp. 2481–2497.

[41] Yisroel Mirsky et al. “Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection”. In: *arXiv preprint arXiv:1802.09089* (2018).

[42] T.M. Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997. ISBN: 9780070428072. URL: https://books.google.com/books?id=xOGAngEACAAJ.

[43] Srinivas Mukkamala, Andrew H Sung, and Ajith Abraham. “Intrusion detection using an ensemble of intelligent paradigms”. In: *Journal of network and computer applications* 28.2 (2005), pp. 167–182.

[44] Reyadh Shaker Naoum, Namh Abdula Abid, and Zainab Namh Al-Sultani. “An enhanced resilient backpropagation artificial neural network for intrusion detection system”. In: *International Journal of Computer Science and Network Security (IJCSNS)* 12.3 (2012), p. 11.

[45] Salima Omar et al. “Machine Learning Techniques for Anomaly Detection: An Overview”. In: *International Journal of Computer Applications* 79 (Oct. 2013). DOI: 10.5120/13715-1478.

[46] Vern Paxson. “Bro: a system for detecting network intruders in real-time”. In: *Computer networks* 31.23-24 (1999), pp. 2435–2463.

[47] Clifton Phua et al. “A comprehensive survey of data mining-based fraud detection research”. In: *arXiv preprint arXiv:1009.6119* (2010).

[48] Marco AF Pimentel et al. “A review of novelty detection”. In: *Signal Processing* 99 (2014), pp. 215–249.

[49] Leonid Portnoy. “Intrusion detection with unlabeled data using clustering”. PhD thesis. Columbia University, 2000.

[50] Mayu Sakurada and Takehisa Yairi. “Anomaly detection using autoencoders with nonlinear dimensionality reduction”. In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. ACM. 2014, p. 4.

[51] Gregor Schaffrath and Burkhard Stiller. “Conceptual integration of flowbased and packet-based network intrusion detection”. In: *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer. 2008, pp. 190–194.

[52] Bernhard Schölkopf et al. “Support vector method for novelty detection”. In: *Advances in neural information processing systems*. 2000, pp. 582–588.

[53] Mansour Sheikhan and Zahra Jadidi. “Flow-based anomaly detection in high-speed links using modified GSA-optimized neural network”. In: *Neural Computing and Applications* 24.3-4 (2014), pp. 599–611.

[54] Taeshik Shon and Jongsub Moon. “A hybrid machine learning approach to network anomaly detection”. In: *Information Sciences* 177.18 (2007), pp. 3799–3821.

[55] Johan Sigholm and Massimiliano Raciti. “Best-effort Data Leakage Prevention in inter-organizational tactical MANETs”. In: *IEEE Military Communications Conference (MILCOM 2012), 29 Oktober 2012-1 November 2012, Orlando, Florida, USA*. IEEE Communications Society. 2012, pp. 1143–1149.

[56] Robin Sommer and Vern Paxson. “Outside the closed world: On using machine learning for network intrusion detection”. In: *IEEE Symp. S&P*. IEEE. 2010.

[57] Sui Song and Zhixiong Chen. “Adaptive network flow clustering”. In: *Networking, Sensing and Control, 2007 IEEE International Conference on*. IEEE. 2007, pp. 596–601.

[58] Anna Sperotto and Aiko Pras. “Flow-based intrusion detection”. In: *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE. 2011, pp. 958–963.

[59] Anna Sperotto et al. “An Overview of IP Flow-based Intrusion Detection.” In: *IEEE Communications Surveys and Tutorials* 12.3 (2010), pp. 343– 356.

[60] Nidhi Srivastav and Rama Krishna Challa. “Novel intrusion detection system integrating layered framework with neural network”. In: *2013 3rd IEEE International Advance Computing Conference (IACC)*. IEEE. 2013, pp. 682–689.

[61] Sutapat Thiprungsri and Miklos A Vasarhelyi. “Cluster analysis for anomaly detection in accounting data: An audit approach”. In: (2011).

[62] Fredrik Valeur et al. “Comprehensive approach to intrusion detection alert correlation”. In: *IEEE TDSC* (2004).

[63] Philipp Winter, Eckehard Hermann, and Markus Zeilinger. “Inductive intrusion detection in flow-based network data using one-class support vector machines”. In: *2011 4th IFIP international conference on new technologies, mobility and security*. IEEE. 2011, pp. 1–5.

[64] Dit-Yan Yeung and Yuxin Ding. “Host-based intrusion detection using dynamic and static behavioral models”. In: *Pattern recognition* 36.1 (2003), pp. 229–243.

[65] Mahmood Yousefi-Azar et al. “Autoencoder-based feature learning for cyber security applications”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 3854–3861.

[66] Chunlin Zhang, Ju Jiang, and Mohamed Kamel. “Intrusion detection using hierarchical neural networks”. In: *Pattern Recognition Letters* 26.6 (2005), pp. 779–791.