Detecting Network Intrusions from Authentication Logs

by

Haibo Bian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2019

Author's Declaration

Abstract

Recently, network infiltrations due to *advanced persistent threats (APTs)* have grown significantly, resulting in considerable loses to businesses and organizations. APTs are stealthy attacks with the primary objective of gaining unauthorized access to network assets. They often remain dormant for an extended period of time, which makes their detection challenging. In this thesis, we leverage *machine learning (ML)* to detect hosts in a network that are a target of an APT attack. We evaluate a number of ML classifiers to detect susceptible hosts in the Los Alamos National Lab (LANL) dataset. We (i) leverage graph-based features extracted from multiple data sources *i.e.*, network flows and host authentication logs, (ii) use feature engineering to reduce dimensionality, (iii) explore balancing the training dataset using numerous over- and under-sampling techniques, (iv) compare our model to the state-of-the-art approaches that leverage the same dataset, and show that our model outperforms them with respect to prediction performance and overhead, and (v) perturb the attack patterns of LMs, study the influence of change in attack frequency and scale on classification performance, and propose a solution for such adversarial behavior.

Acknowledgements

I would like to thank my supervisor Professor Raouf Boutaba for his help in this project. His support made my research life resourceful and productive. It was a good experience working with him.

Special thanks to Dr. Mohammad A. Salahuddin for his continuous help in the writing, Tim Bai for his advice in the experiments, Professor Noura Limam for her guidance, Shihabur Rahman Chowdhury for his help in proofreading and presentation preparation, and Abbas Abou Daya for his inspiration and suggestions during the initial stage of this work.

Great thanks to my family for their support during my stay here.

Dedication

This is dedicated to my family.

Table of Contents

List of Tables viii

List of Figures x

1 Introduction 1

1.1 Contributions 4

1.2 Thesis Organization 4

2 Background 6

2.1 APT and Lateral Movement 6

2.2 Lateral Movement Detection 7

2.2.1 Anomaly Detection 7

2.2.2 Flow-Based LM Detection 8

2.2.3 Host-Based LM Detection 9

2.2.4 Hybrid LM Detection 11

2.3 Adversarial Learning 11

3 Methodology 13

3.1 Dataset 13

3.1.1 Characteristics 13

3.1.2 Balancing 16

3.2 Feature Extraction 18
3.3 ML Techniques 22
3.4 Evaluation Metrics 22

4 Evaluation **24**
4.1 Environment 24
4.1.1 Hardware 24
4.1.2 Software 24
4.2 Results 24
4.2.1 Feature Selection 24
4.2.2 Ensemble Learning 27
4.2.3 Balancing the Dataset 28
4.2.4 Comparative Analysis 30

5 Adversary Study **33**
5.1 Perturbing Number of TAs 34
5.2 Perturbing Attack Frequency 40
5.3 Adversarial Learning 46

6 Conclusion and Future Work **50**
6.1 Conclusion 50
6.2 Future Work 51

References **52**

APPENDICES **58**

A Appendix **59**
A Feature Index Table 59
B Build Authentication Dictionary 60
C Generating Features from Dictionaries 65

List of Tables

2.1 Summary of APT stages 7

3.1 Most significant features extracted from authentication logs 18

3.1 Most significant features extracted from authentication logs (contdd.) 19

4.1 ML performance using flow- and authentication-based features (35 features) 25

4.2 ML performance using authentication-based feature set (29 features) 26

4.3 Pearson correlation matrix for most correlated authentication-based features 26

4.4 ML performance on reduced authentication-based feature set (17 features) 27

4.5 Ensemble learning using majority voting 27

4.6 Ensemble learning using weighted voting, prioritizing stand-alone ML performance 28

4.7 Over-sampling with different algorithms (17 features) 29

4.8 Under-sampling with different algorithms (17 features) 29

4.9 Comparing different algorithms (17 features) 30

4.10 TA detection using stand-alone RF and cross-validation versus ([15], [31]) . 30

4.11 Robustness of TA detection using stand-alone RF versus ([15], [31]) 31

4.12 Robustness of TA detection using stand-alone RF versus ([15], [31]) on a reduced training dataset 32

5.1 Decrease in number of TAs 37
5.2 Decrease in number of TAs and classifier threshold according to F1 score . 38
5.3 Decrease in number of TAs and *Ratio*, using sparse related features only . 38
5.4 Increase in number of TAs, using uniform threshold 40
5.5 Increase in number of TAs, using sparse features only 40
5.6 Decrease in malicious authentication events frequency with different *Ratio* 43
5.7 Decrease in malicious authentication events frequency with different *Ratio*, using sparse features only 44
5.8 Increase in malicious authentication events frequency with different *Ratio* . 45
5.9 Increase in malicious authentication events frequency with different *Ratio*, using sparse features only 46
5.10 Decrease in number of TAs, adding synthetic TAs to training dataset ($Ratio = 0.6$) 48
5.11 Increase in malicious authentication events frequency, adding synthetic TAs to training dataset ($Ratio = 0.6$) 48
5.12 Decrease in number of TAs, adding synthetic TAs to training dataset ($Ratio = 0.1$) 48
5.13 Increase in malicious authentication events frequency, adding synthetic TAs to training dataset ($Ratio = 0.1$) 49

List of Figures

1.1 Number of Hosts vs. Number of Events: Random sample of hosts from the original LANL dataset and corresponding authentication events. 2

1.2 Number of Hosts vs. Feature Extraction Time: Random sample of hosts from five days in the original LANL dataset and features from corresponding authentication events. 3

3.1 Redteam activities distribution 14

3.2 Authentication events distribution 14

3.3 Network flow distribution 15

3.4 Redteam network flow distribution 16

3.5 Graph representation of authentication events 20

4.1 ROC for robustness in TA detection using stand-alone RF vs. ([15], [31]), with days 13, 14 and 15 for training, and day 9 reserved for testing 32

5.1 Attacker changes probing pattern: (a) the original attack, (b) increasing attack frequency (c) increasing the attacking scale (*i.e.*, attack more TAs). Note that arrow labels correspond to authentication events. 34

5.2 Example of increase in number of TAs with $Ratio = 2$: (a) original attack, (b) attacking more TAs, for each new host (*i.e.*, victim VC_i for corresponding original TA_i) map to it the original attack a_i on TA_i *i.e.*, a'_i. Note, attack a_i on TA_i is a set of authentication events. 35

5.3 AUC with decrease in number of TAs and *Ratio* 36

5.4 AUC with decrease in number of TAs and *Ratio* (zoom view) 37

5.5 AUC with increase in number of TAs and *Ratio* 39

5.6 Example of increase in attack frequency with *Ratio* = 2: (a) the original attack, (b) for each attack a_i on the original TA_i, we add randomness to its attack time, which results in a'_i that is simulated on TA_i. Note, attack a_i on TA_i is a set of authentication events. 41

5.7 AUC with decrease in attack frequency 42

5.8 AUC with decrease in attack frequency (zoomed view) 43

5.9 AUC with increase in attack frequency 45

5.10 Example of generating synthetic TAs by decreasing the number of TAs with *Ratio* = 0.6: (a) the original attack, (b) randomly choose TA'_i from the original TA_i. Add randomness to each attack a_i's attack time, which results in a'_i that is simulated on TA'_i. Note, attack a_i on TA_i is a set of authentication events. 47

Chapter 1

Introduction

Cyber attacks have been growing in sophistication, resulting in considerable damage to businesses. They not only result in financial losses, but also impact customer trust and churn. There has been an increasing trend in cyber attacks in recent years. Typically, an attack initiates by compromising several hosts or user accounts within a network, and leaves backdoors to gain persistent access to internal assets. This type of attack is commonly known as an advanced persistent threat (APT). According to Kaspersky Lab, an APT campaign in 2019 affected over a million users who installed the ASUS Live Update utility [25]. Similarly, a cryptocurrency exchange firm, DragonEx, announced in 2019 that it has suffered USD 7.09 million in losses due to an APT attack [52]. Therefore, it is imperative to defend businesses against APT-assisted network intrusions.

Lateral movement (LM) is a crucial phase in an APT attack, which follows after an attacker has gained persistent access to certain network resources (*e.g.*, servers, end-hosts). The goal of LM is to infiltrate other resources and gain higher privileges inside the target network. This is typically achieved by performing credential stealing or vulnerability exploitation on already compromised hosts. Interestingly, 50%–90% of employees have access to data that they no longer need [50]. This is primarily due to poor security practises, such as the violation of the least privilege principle [49], which increases the likelihood of an attacker penetrating the crucial network assets via LM. Therefore, it is vital to detect LM at an early stage.

As opposed to the traditional detection of successful intrusions [2], an alternative is to pro-actively identify covert signs of LM. This can potentially generate alarms even before a successful intrusion has occurred, leading to LM detection during early exploration. After acquiring footprints of such behaviour, administrators can get insights into the attack strategy. They can also identify system vulnerabilities, which can help alleviate future attacks. However, unlike hosts that act as proxies during the attack, newly compromised or vulnerable hosts are fairly dormant and leave minimal footprint (*e.g.*, events in authentication logs). Furthermore, in large enterprises with thousands of hosts, it is unlikely that an infiltration will compromise the majority of hosts. Typically, the number of compromised hosts will be minuscule in comparison to the network size, resulting in sparse malicious activities. These issues make early detection of LM challenging.

Early detection of LM can be addressed by: (i) tagging the malicious host events, or (ii) tagging the target assets (TA). However, the stealthiness and sparseness of malicious events can make the first strategy a difficult endeavor. Though, crafting discriminative features for each event can achieve high recall, it comes at a high computational overhead. For instance, Fig. 1.1 shows that the number of events increases with the network size. Similarly, Fig. 1.2 shows that feature extraction time for event tagging using the method described in [31] significantly increases with the number of hosts in the network. This drawback makes the first strategy unscalable for very large networks. Furthermore, for complex network infrastructure with sporadic events, tagging individual events can also result in a high number of false positives. In comparison, tagging TAs reduce computational overhead. With carefully crafted features from sparse events, it is possible to achieve a high precision in detection performance (*cf.*, Chapter 4). We focus on the second strategy for early detection of LM by leveraging host authentication logs.

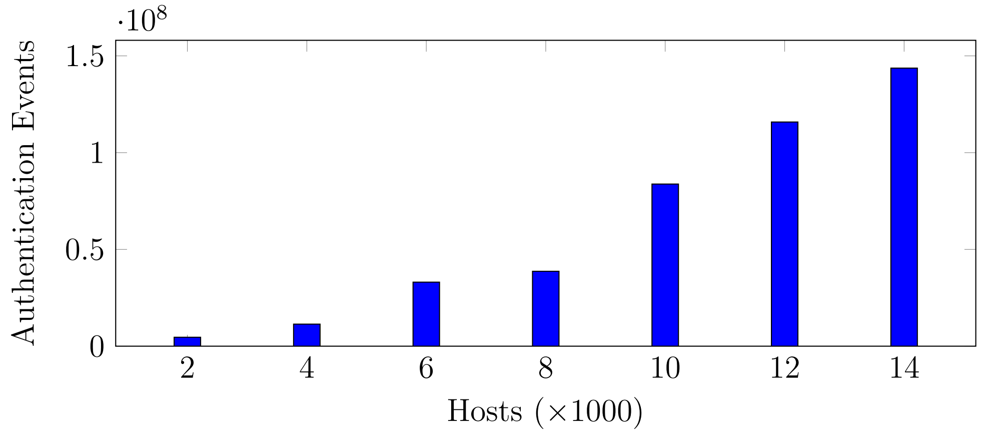

Figure 1.1: Number of Hosts vs. Number of Events: Random sample of hosts from the original LANL dataset and corresponding authentication events.

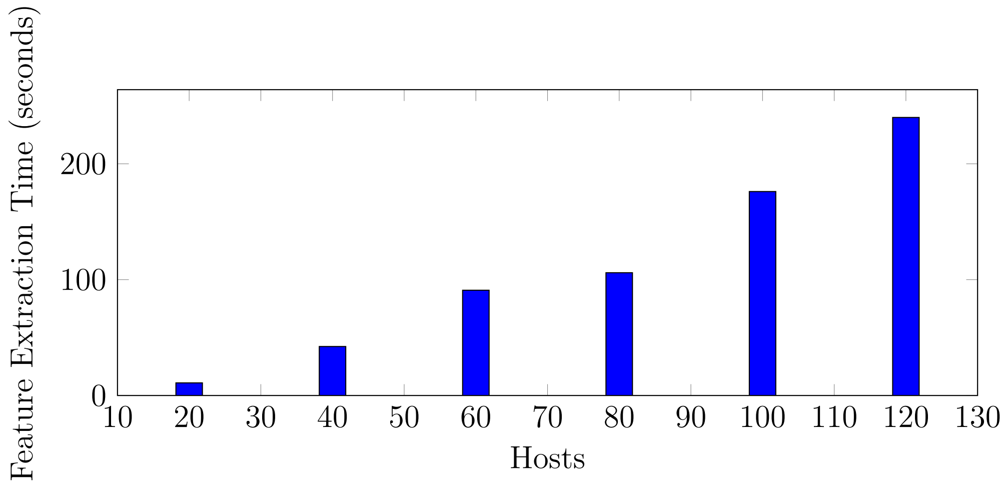

Figure 1.2: Number of Hosts vs. Feature Extraction Time: Random sample of hosts from five days in the original LANL dataset and features from corresponding authentication events.

Anomaly-based methods are widely used for intrusion detection. These methods first establish a baseline of normal system behavior and model a decision engine. The decision engine determines and alerts any divergence or statistical deviations from the norm as a threat. Machine learning (ML) [5, 10] is an ideal technique to automatically establish the normal behavior of a system. However, an important step prior to training a ML model is feature extraction. These features act as discriminators for learning and inference, and increase the accuracy of ML models. The most commonly employed features in intrusion detection are either network flow-based (*e.g.*, number of packets, direction, packet size and inter-arrival statistics) or host event-based (*e.g.*, authentication type, authentication frequency, and user names used during authentication). However, these features do not completely capture the host communication patterns that may expose additional aspects of malicious behavior. Graph-based features, derived from flow-level or event-level information to reflect the true behaviour of hosts, are an alternative that overcome this limitation.

The distribution of the dataset can also severely influence ML performance. For example, in the case of an imbalanced dataset (*e.g.*, sparse malicious host events versus benign events), the ML techniques are more likely to classify new data to the majority class. Though, balancing the dataset can alleviate this issue, it may sabotage ML performance by impacting graph-based features (*cf.*, Chapter 4). Moreover, ML-based intrusion detection systems can be vulnerable to adversarial perturbations [4]. Adversaries can introduce variations in their attack patterns to evade detection (*e.g.*, by varying the frequency and/or scale of the attack, adding randomness to their behaviour, *etc.*) In some cases, introducing noise and variation in the training data can help improve classification performance, making the system more robust to adversarial attacks. (*cf.*, Chapter 5).

1.1 Contributions

In this thesis, we propose a novel approach for anomaly-based early detection of LM. Our main contributions are:

- We leverage ML for identifying TAs to facilitate early detection of LM. In this respect, we evaluate numerous supervised ML techniques and their ensemble, and compare them in classification performance and overhead.
- We employ graph-based features using real datasets from the Los Alamos National Lab (LANL) [32]. We explore features that are extracted from multiple data sources *i.e.*, network flows as well as host authentication logs, and employ feature engineering to reduce dimensionality.
- Due to the highly imbalanced nature of the LANL dataset, we evaluate various over- and under-sampling techniques, and explore their impact on ML performance.
- We compare our approach to state-of-the-art approaches that leverage the LANL dataset for detecting LM. We show that our approach outperforms the other approaches with respect to detection performance and overhead.
- We perturb the attack patterns of LMs and study the influence of change in attack frequency and scale on classification performance. We study different methods to sample the dataset and mitigate the influence of attack pattern changes.

1.2 Thesis Organization

The rest of the thesis is organized as follows.

- Chapter 2 highlights the most recent related works on LM detection.

- Chapter 3 discusses the characteristics of the LANL dataset, delineates the explored sampling algorithms, exposes feature extraction and selection, and presents the ML techniques employed for TA detection.
- Chapter 4 discusses the results of our evaluation and comparison to the state-of-the-art approaches for LM detection.
- Chapter 5 explores the influence of attack pattern change and mitigates the influence by balancing the training dataset.
- Chapter 6 concludes with a brief summary and outlines future research directions.

Chapter 2

Background

In this chapter, we first introduce APT and LM in Section 2.1. Then we expose the existing approaches for LM detection. Sections 2.2.2 and 2.2.3 summarize works on flow-based and host-based LM detection, respectively. Section 2.2.4 describes detection methods with evidence from multiple data sources.

2.1 APT and Lateral Movement

In an APT, attackers gain access to systems or networks and reside there in for an extended period of time. Different from traditional intrusions, APTs have several distinct characteristics: (i) clear objectives, (ii) highly organized and well resourced, (iii) long term persistence with repeated attempts, and (iv) stealthy and zeal to stay undetected [16]. Due to these characteristics, APTs are challenging to detect. Nevertheless, APTs tend to have distinct characteristics, that lead to several common stages in an attack. A summary of the typical steps performed by an APT is presented in Table 2.1. During Reconnaissance, attackers collect information about the target from various resources. They may be sophisticated and gather information from the organization websites, multimedia, and social networks. Then by tricking the employees to clicking on crafted URLs or opening malicious attachments, attackers deliver the malware onto the target hosts. With the malware successfully installed on the victim host, a command and control channel is used to manage the victim from the master server. After moving laterally and compromising further hosts, the attacker finds the valueable assets and steals them through data exfiltration.

LM is an approach used by adversaries to systematically explore the network to access valuable assets. As a crucial stage in APT, LM normally lasts over the longest span of time. Attackers gain more information about the system by scanning the network and compromising more hosts. To explore the internal network as well as remain undetected, attackers perform this stage slowly, leaving minimum footprints. They often reduce the attack frequency and utilize legitimate system tools [30]. Mixed in a large volume of benign activities, identifying LMs can be challenging.

Table 2.1: Summary of APT stages

Step	**Summary**
Reconnaissance	gather information about the target organization
Delivery	deliver exploits to the target
Initial Intrusion	get first unauthorized access to the target
Command and Control (C2)	use C2 to control compromised hosts
Lateral Movement	move inside network and expand control
Data Exfiltration	steal sensitive data

2.2 Lateral Movement Detection

Two attributes of LM make its detection a challenging problem. First, a plethora of possible attacking techniques (Pass the Hash, Remote Desktop Protocol, Remote services *etc.*) can be used during the lateral movement stage [18]. They leave different traces on either the network, the hosts, or both. Second, to remain undetected, attackers launch their attacks slowly and infrequently, as a result, their traces are mixed with the copious benign activities. Typically, researchers tackle the first problem by using anomaly detection. As opposed to detecting states (number of bytes received, CPU usage, memory usage, *etc.*) caused by the attack, they detect new attacks that result in an undefined state. For the second problem, different methods collect all evidence available, seeking traces of LMs from both the host and the network. They also balance the dataset with well-known or self-defined methods.

2.2.1 Anomaly Detection

Anomaly detection has been extensively used for network intrusion detection [12] [24]. As opposed to methods targeting characteristics of individual attacks, it detects activities aberrant from normal forms, which makes the method robust to new attacks. After building the knowledge of the normal behavior, the system assigns abnormal scores to new activities, which are then classified accordingly. Works in this area can be classified into three categories: statistic-based [22] [59], knowledge-based and ML-based anomaly detection [47] [37].

Statistic-based anomaly detection systems (SB-ADS) build the baseline from historical data, such as the frequency of network events, kind of protocols used, and the number of destination hosts contacted. By comparing the profiles of historical data and that of new incoming events, SB-ADS decides the degree of irregularity with a score. Over a span of time, statistic-based systems can efficiently detect malicious activities with well-defined normal profiles. However, they are vulnerable when the normal profile is contaminated. Attackers can interfere during the training phase, which can cause attacking events to be classified as benign. On the other hand, knowledge-based ADS infers the legitimacy of events according to pre-defined rules. However, benign patterns that have not been anticipated, can be misclassified in this approach. Besides, composing the rules itself is difficult and time consuming [24]. Compared to the previous two methods, ML-based approach can adapt to behavioral changes. It resembles SB-ADS in theory, but with new labeled data and properly selected models, ML-based approach can improve its performance progressively. In this work, we focus on leveraging ML-based approach to capitalize on its robustness to new attack patterns.

2.2.2 Flow-Based LM Detection

As attackers move laterally in the intranet, they leave footprints on the network regardless of the attacking strategy. A significant body of work exists in the research literature that identifies LM using flow-based features.

Micro *et al.* [39] propose a framework to detect hosts that are involved in data exfiltrations. They first calculate an abnormal score based on the outgoing traffic, then rank the hosts based on this metric. The top K hosts are inspected for verification. However, the detection is limited to APTs that include data exfiltration. Furthermore, this entails an overhead for administrators to inspect individual hosts. Ullah [55] proposes a system that leverages the BRO network analyzer. It extracts features from network traffic and uses k-nearest neighbor (k-NN) classifier. Their system is constrained with Server Message Block hosts, which is not general enough to cover broad LM activities. Both of these works are limited to a certain type of LM attack. In comparison, our approach is agnostic to the employed LM strategy.

Bhasin *et al.* [7] propose to detect traffic anomalies using the Jaccard Similarity Coefficient and Clustering technique. Their system can detect LM-related traffic given limited hosts and applications. However, their system is built on the assumption that most normal traffic is similar, which does not hold true for complex network topologies with hosts running heterogeneous applications. As a result, their work is restricted to highly analogous network infrastructures.

Dahiya *et al.* [20] apply Canonical Correlation Analysis and Linear Discriminant Analysis (LDA) for reducing feature dimensionality. The authors apply these methods on the UNSW-NB15 [42] dataset and compare seven different ML techniques. They show that random tree yields the best result, i.e., 86.46% and 86.1% in accuracy and precision, respectively.

Ghafir *et al.* [26] construct a system that detects intrusion in three phases: (i) threat detection, (ii) alert correlation, and (iii) attack prediction. In the first phase, eight different modules individually generate alerts for different attacking strategies. In the second phase, alerts are filtered, clustered and indexed. Finally, a prediction module is trained and tested based on the output from the second phase. Their system achieves 81.8% and 4.5% in true positive rate and false positive rate, respectively.

Current flow-based LM detection frameworks cover detection from different aspects. However, they are either targeting highly constrained scenarios or detecting LM with a focus on a certain attacking pattern. In contrast, our work detects LM's exploring behaviour, regardless of the attacking strategy or pattern.

2.2.3 Host-Based LM Detection

Even though flow logs can contain traces of LM, the complex network topology and diverse applications introduce noise in flow-based systems. On top of that, widely adopted encryption protocols, such as TLS [48], limit the amount of information available from the network traffic. The encryption endpoint, however, reveals fine-grained information, such as running processes and user accounts. These insights can enable host-based systems to differentiate between benign and malicious behavior. Thus, there are numerous works that leverage host-level information for LM detection.

Liu *et al.* [38] propose a ranking system to detect targeted hosts. The authors first construct a connection graph from Windows security events generated by Kerberos service ticket requests. Then, they rank the hosts according to path-rate score, which reflects the rarity of a path in the network. To reduce the false positives, the authors construct Remote File Execution Detector to filter out benign behaviours. Their system can detect all remote file execution related LMs within a subset of the dataset.

Chen *et al.* [15] leverage features from multiple data sources to identify LM. They utilize rudimentary graph-based features based on host communication, while employing autoencoder to improve feature extraction. To address imbalance in the LANL dataset, the authors propose a custom under-sampling technique. They employ k-NN and achieve an average of 91.3% precision in LM detection. However, their evaluation is limited to the k-NN classifier.

Bohara *et al.* [9] propose an unsupervised approach to detect malicious LM. They employ the LANL dataset and inject artificial attacks into the original dataset, instead of using redteam events (available in the dataset). However, these simulated attacks may not depict behavior of real attacks in enterprise networks. Their LM activity simulation follows the susceptible-infected-susceptible virus spread model [56]. The authors extract features from host communication graphs, while principal component analysis (PCA) is used to correlate different features. For detection, they propose a combination of two different detectors to enhance performance. The first detector uses PCA and k-means, while the second one employs PCA and extreme value analysis. This combination achieves an 88.7% true positive rate.

Chawla *et al.* [13] apply Convolutional Neural Network (CNN) with Gated Recurrent Unit (GRU) on the ADFA-LD [19] dataset. The CNN layers can capture local correlation of structures in the sequences and execute in parallel, which improves the performance. The Recurrent Neural Network (RNN) with GRU at the same time can learn sequential correlations from those higher-level features. Their system can reach an Area Under the Curve (AUC) score of 0.81 after training on normal sequences. However, a well balanced dataset can not represent real attacks. In reality, APTs are stealthy, which will result in a highly unbalanced trace, as evident in the LANL dataset. Failing to test against imbalanced data undermines the classifier's ability against real attacks.

Tuor *et al.* [54] and Brown *et al.* [11] propose RNN for log level anomaly detection. Tuor *et al.* introduce a language modeling framework for generic log level anomaly detection, while Brown *et al.* extend a previous framework and focus on developing RNN models with attention mechanism. These efforts do not employ feature engineering, but rather the ML models directly leverage tokenized log lines. They achieve AUC of 0.98 and 0.99, respectively. However, AUC is impacted when the dataset is highly imbalanced. In contrast, our approach operates on the host level, whereas the aforementioned approaches detect on the log level.

2.2.4 Hybrid LM Detection

Both host-based and flow-based LM detection systems have their advantages and disadvantages. However, flow-based approaches have access to limited information, since the packet payload is mostly encrypted. Host-based approaches can leverage more information from the end system, but may not detect LMs that leverage stealthier vulnerabilities [41]. Thus, a hybrid system that utilizes both host- and flow-based evidence can benefit from the advantages of both approaches.

Several other works [34, 17, 1] propose a hybrid intrusion detection system. Kim *et al.* [34] propose a hierarchical approach that decomposes normal training data into smaller subsets using decision tree (DT) and leverage one-class support vector machine (SVM) for each subset. Chitrakar *et al.* [17] propose a similar approach, where the training data is split into different clusters using k-medoids, followed by naïve bayes for further attack classification. Agarwal *et al.* [1] normalize entropy of network features using a custom algorithm and leverage SVM for attack classification. However, none of the aforementioned approaches show both precision and recall in their evaluation. But from the partial metrics available we can detect their inferior performance in recall. They combine multiple techniques for classification, but none of them explore data from different sources. In contrast, we explore features extracted from multiple data sources in an effort to improve classification performance.

Our work leverages trial-and-error and observations from the LANL dataset. The data structure used (*i.e.*, bipartite graph) is inspired from Kaiafas *et al.* [31]. They construct a bipartite graph to extract graph-based features and employ an ensemble of ML models to improve classification performance. However, the authors only perform k-fold cross-validation and do not evaluate the robustness of their ML models to unseen data. This is crucial to ensure the detection of zero-day APTs. We highlight this limitation in Chapter 4.

2.3 Adversarial Learning

ML has been extensively explored for cybersecurity. However, research shows that crafted adversarial samples can hamper the performance of ML [44, 45, 51, 58]. Xu *et al.* [58] propose a generic method to identify evasive samples. After the test against two recent PDF malware detectors, their system successfully evades detection with 100% success rate. Biggio *et al.* [8] experiment with a gradient-based approach to evade detection and show that popular classification algorithms, such as SVM and neural networks are vulnerable.

Anderson *et al.* [3] leverage generative adversarial networks to thwart the performance of a deep learning-based detector, which detects the use of Domain Generation Algorithm (DGAs). Most of the works in adversarial learning are geared towards deep learning approaches. Our approach is based on random forest, which is a well known and widely used traditional learning algorithm. We perform experiments to explore the influence of adversarial samples on our approach for TA detection (*cf.*, Chapter 5).

Chapter 3

Methodology

3.1 Dataset

3.1.1 Characteristics

The LANL dataset contain logs from multiple data sources, including authentication log, flow log, DNS log, and process log. We explore the authentication and flow logs for TA detection during LM.

Authentication Log

This log is composed of over 450 million authentication events from Windows-based desktop computers, spanning 58 days. Among these events there are 749 redteam compromise events, distributed in the first 30 days of the dataset, as depicted in Fig. 3.1 and Fig. 3.4. We leverage data in this time frame, which consists of about 230 million events from 14,582 benign hosts and 299 redteam related hosts. However, the malicious activities are a very small fraction of the overall activities in the LANL dataset. Fig. 3.2 shows the daily events distribution. In comparison to Fig. 3.1, it also reveals that malicious activities are sparse, both in general and on a daily basis.

We do not consider local redteam authentication events *i.e.*, malicious events where the source and destination hosts are the same. The behavior of an attacker that performs malicious activity within a physical machine tends to be quite different. Such an attacker has access to the physical interfaces of the host, hence their attack strategy and behaviour can be very sophisticated. Evaluating such behavior is out of scope for this thesis. Nevertheless, we capitalize on the number of infrequent events. In total, there are 8,941 hosts involved in 41,400 authentication events that occur only once in the dataset. Out of the 295 TAs, 280 are involved in such events. Therefore, considering the event infrequency *i.e.*, *sparseness*, can potentially facilitate the detection of TAs.

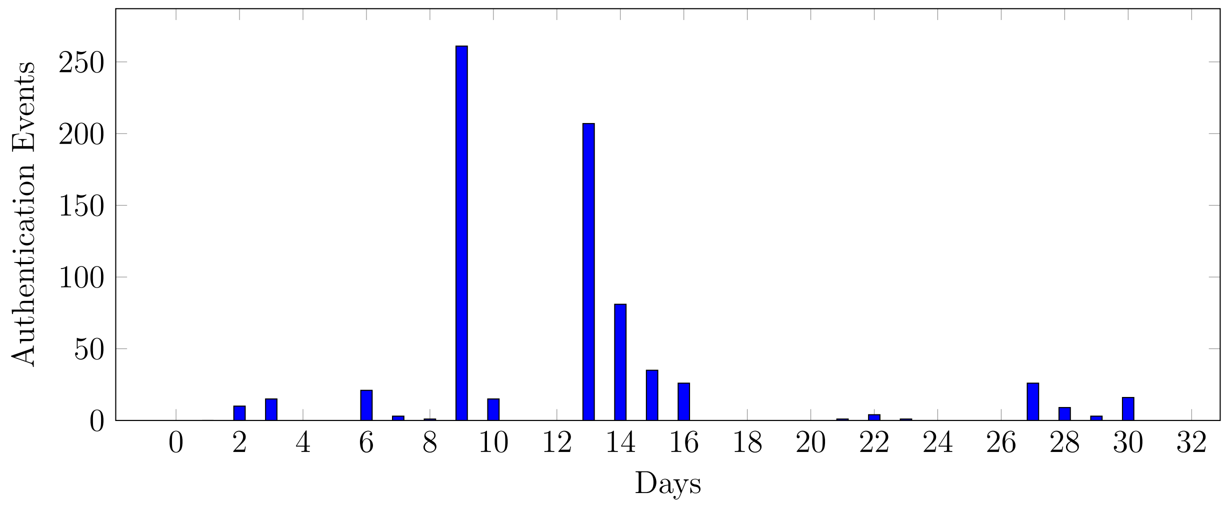

Figure 3.1: Redteam activities distribution

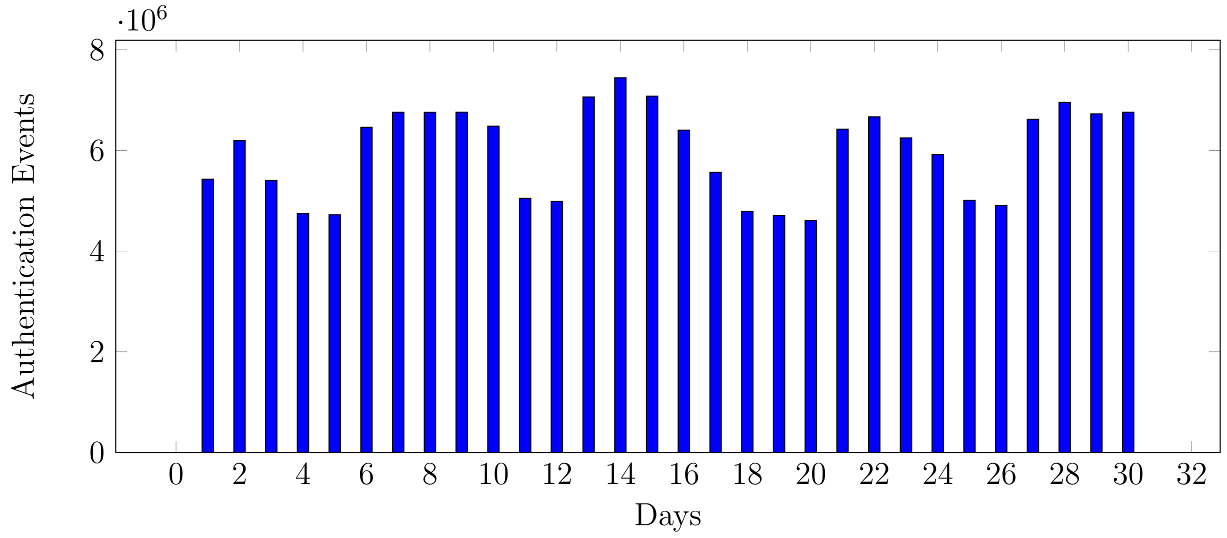

Figure 3.2: Authentication events distribution

Flow Log

This log contains flow events collected from the central routers in the network. It spans 30 days with a total of 129 million flows, as shown in Fig. 3.3. There is a noticeable change in distribution from day 16, which indicates a major change in the system. It is also worth mentioning that flows that do not go through the central routers will not be recorded. As a result, out of the 14,881 hosts in the authentication log, only 3,456 hosts have corresponding flow data. Also, due to a potential misconfiguration of internal network routers, the flow data collection completely stops after day 29 [33]. Fig. 3.4 shows the distribution of redteam-related flows. Compared to the entire dataset, redteam flows are minuscule, which sum up to 45 million in total. Thus, the flow log is also imbalanced.

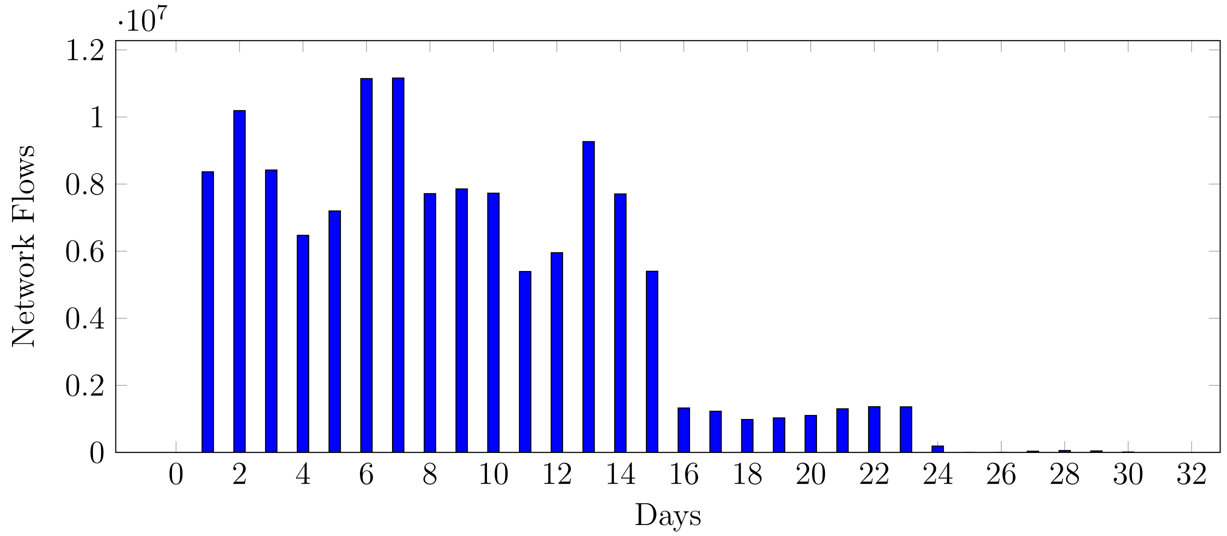

Figure 3.3: Network flow distribution

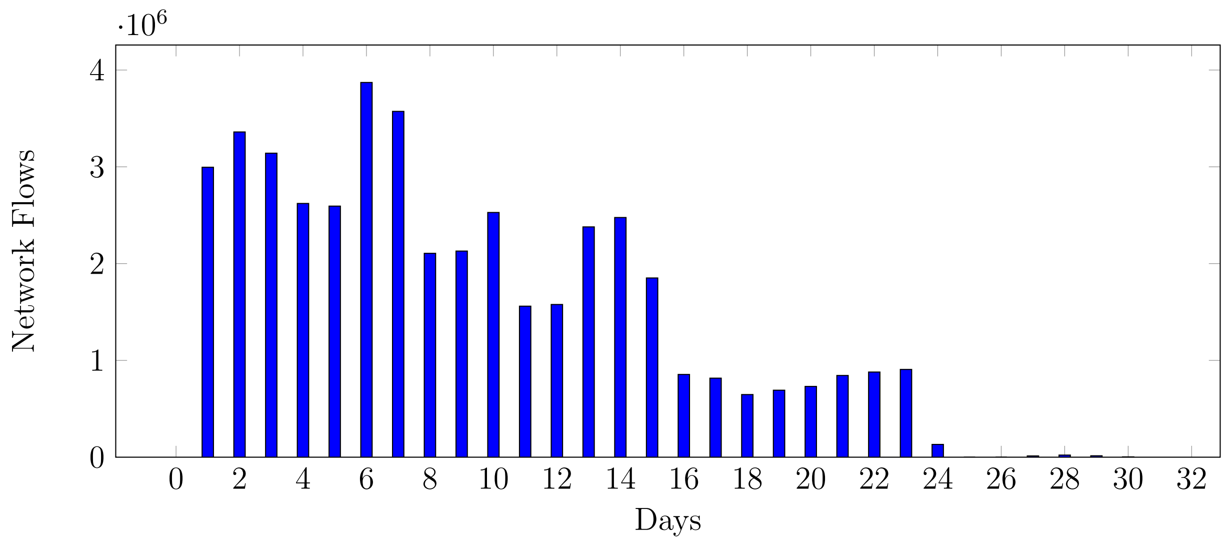

Figure 3.4: Redteam network flow distribution

3.1.2 Balancing

Sampling algorithms are employed when a dataset is highly imbalanced. An imbalanced dataset can result in a classifier that is biased on the majority class, due to the nature of the training procedure. The sampling algorithms used to alleviate this issue can be classified into two categories, under-sampling and over-sampling. While under-sampling approaches balance the dataset by reducing the data points in the majority class, the over-sampling approaches increase the data points in the minority class. Therefore, the under-sampling algorithms are known to inherently lose critical information, while the over-sampling algorithms suffer from over-fitting [6]. However, a potential advantage of under-sampling is the reduced computational overhead. On the other hand, some classifiers have the capability to overcome the over-fitting due to over-sampling.

We explore different algorithms from both categories for balancing the LANL dataset. The first algorithm is random under-sampling (RUS), which randomly removes samples from the majority class. The second algorithm is condensed nearest neighbour (ConNN), an under-sampling algorithm based on k-NN [27]. This algorithm keeps all samples in the minority class and uses 1-NN classifier to determine whether to retain the data point in the majority class or not. The next algorithm is Repeated Edited Nearest Neighbours (RENN), which implements multiple iterations of Edited Nearest Neighbours (ENN) [57]. For the over-sampling algorithms, we start with random over-sampling (ROS), followed by the well-known synthetic minority over-sampling technique (SMOTE) [14]. SMOTE over-samples data points by creating their synthetic counterparts. This is achieved by computing a vector between a data point and one of its neighbours. Another over-sampling algorithm is the adaptive synthetic (ADASYN) [28]. ADASYN also leverages k-NN to adaptively generate synthetic data.

We employ the above sampling algorithms after feature extraction. This is primarily because applying them directly on the authentication log can sabotage the purity of graph-based features. For example, all authentication events pertaining to a username may get eliminated due to under-sampling. Similarly, over-sampling without considering the diversity of hosts in the dataset may result in emphasizing a single type of host. We study the influence of these sampling algorithms on TA detection in Chapter 4.

3.2 Feature Extraction

Table 3.1: Most significant features extracted from authentication logs

Feature	Definition
$ID_{usr}(dst_j)$	The count of unique *username* used to logon to dst_j
$ID_{src}(dst_j)$	The count of unique *source* hosts that logon to dst_j
$ID_{(usr,src)}(dst_j)$	The count of unique *(username, source)* pairs that logon to dst_j
$IDAF_{usr}(dst_j)$	The average over all *username* of $AVG_{dst_j}(username)$, where $AVG_{dst_j}(username)$ is the number of times *username* is used to logon to dst_j divided by the number of days *username* used to logon to dst_j
$IDAFSTD_{usr}(dst_j)$	Standard deviation of $AVG_{dst_j}(username)$
$IDS_{usr}(dst_j)$	The sum over all *username* of $SF(*, username, dst_j, \theta, \beta)$, where SF is exposed in Algorithm 1
$IDS_{src}(dst_j)$	The sum over all *source* of $SF(source, *, dst_j, \theta, \beta)$
$IDS_{(usr,src)}(dst_j)$	The sum over all *(username, source)* pairs of $SF(source, username, dst_j, \theta, \beta)$
$WIDS_{(usr,src)}(dst_j)$	The sum over all *(username, source)* pairs of $SF(source, username, dst_j, \theta, \beta)$ weighted by $ODS_{(usr,dst)}(source)$
$ODS_{usr}(src_i)$	The sum over all *username* of $SF(src_i, username, *, \theta, \beta)$
$ODS_{dst}(src_i)$	The sum over all *destination* of $SF(src_i, *, destination, \theta, \beta)$
$ODS_{(usr,dst)}(src_i)$	The sum over all *(username, destination)* pairs of $SF(src_i, username, destination, \theta, \beta)$
$ODAFSTD_{(usr,dst)}(src_i)$	Standard deviation of $AVG_{src_i}(username, destination)$, where $AVG_{src_i}(username, destination)$ is the number of times *username* is used by src_i to logon to *destination* divided by the number of days *username* is used by src_i to logon to *destination*

Table 3.1: Most significant features extracted from authentication logs (contd.)

Feature	Definition
$ODAFSTD_{usr}(src_i)$	The standard deviation of $AVG_{src_i}(username)$, where $AVG_{src_i}(username)$ is the number of times *username* is used in a remote login attempt initiated by src_i divided by the number of days *username* is used by src_i in a remote login attempt
$MSF(dst_j)$	The maximum over all src_i of $ODS_{(usr,dst)}(src_i)$ where $SF(dst_j, src_i, in, \theta, \beta) > 0$
$SUR(dst_j)$	The number of unique *username* used to sparsely logon to dst_j ($i.e.,$ $SF(dst_j,(username,source),in,\theta,\beta)$ >0 for at least half the logon events $(username, source, dst_j)$) divided by the number of unique *username* used to logon to dst_j
$AS(Host_j)$	$MSF(Host_j) * SUR(Host_j)$

We extract a total of 35 features, 29 features from the authentication log and 6 from the flow log. A detailed description of the extracted flow log-based features can be found in our previous work [21]. The 29 authentication-based features are extracted from a graph representation of the authentication events. As the features are primarily based on the in-degree and out-degree of different hosts, we build an authentication graph that is efficient for frequent reference. We first start by building the authentication graph $G = (U, V, E)$, where U represents the hosts that appear as sources in the authentication log, while V represents the hosts that appear as destinations. Edges in E link pairs $(u, v) \in U \times V$ and summarize all authentication events involving u as source and v as destination. Authentication events are inserted in the graph, as shown in Fig. 3.5.

For example, consider an authentication event $e(Day_2, User02, ComPtr10099, ComPtr 4017)$, where Day_2 represents the day when the logon was recorded, $User02$ is the username used in the logon attempt, and $ComPtr10099$ is the source host used by $User02$ to logon to destination host $ComPtr4017$. Assuming that $User02$ was already recorded logging into $ComPtr4017$ from $ComPtr10099$ twice on Day_1, 3 times on Day_n, but never before on Day_2, the event e is added to the edge linking $ComPtr10099$ to $ComPtr4017$ on the graph G with a count of 1, as depicted in Fig. 3.5. Once the graph G is complete, we build dictionaries that are used to extract the features, as described in Appendix A

A high-level description of the authentication-based features is provided below. Table 3.1 further delineates a subset of the authentication-based features.

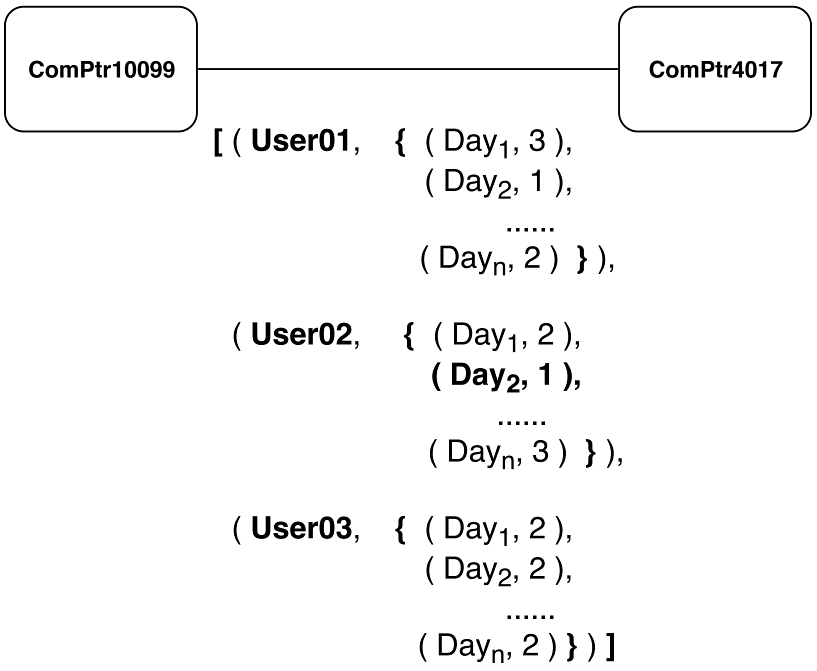

Figure 3.5: Graph representation of authentication events

In-Degree (ID) and Out-Degree (OD): In the early phase of LM, attackers use stolen credential to attempt logging into and eventually compromising other hosts. This will result in the increase of ID of the targeted hosts and OD of successfully compromised ones.

In-Degree-Avg-Frequency (IDAF) and Out-Degree-Avg-Frequency (ODAF): Infrequent malicious authentication events will have little impact on ID/OD in the presence of a much larger number of benign authentication events. Thus, they can be overlooked by the classifier. We consider IDAF, the daily average number of authentication events targeting the host, as well as ODAF, the average number of authentication events originating from the host, and leverage the discriminatory nature of these features.

IDAF-Standard-Deviation (IDAFSTD) and ODAF-Standard-Deviation (ODAF-STD): Sparse malicious authentication logs can be shadowed by regular and repetitive benign logons when calculating IDAF and ODAF. On the other hand, the standard deviation of IDAF will be higher for TAs targeted by a mix of frequent legitimate logons and sparse malicious logons, than non-TAs. Similarly, compromised hosts will have higher ODAFSTD than benign ones.

In-Degree-Sparseness (IDS) and Out-Degree-Sparseness (ODS): In order to capture infrequent events that are likely to be malicious, we introduce a *sparseness function* (SF), as depicted in Algorithm 1. SF considers infrequent events with a specific (combination of) *source host*, *destination host*, or *username*, and assigns a higher score to more infrequent events. This amplifies the impact of such events on graph-based features, which are otherwise largely affected by benign events. IDS and ODS reflect the sparseness of the incoming and outgoing logons, respectively. In this case, SF evaluates the sparseness of these events and amplifies the impact of sparse authentication events when computing the ID of a TA and OD of a compromised host. As sparse malicious logons receive higher SF scores, TAs are expected to have higher IDS than non-TAs, and compromised hosts to have higher ODS than benign ones.

Weighted-In-Degree-Sparseness (WIDS): To distinguish between TAs and non-TAs with comparable IDS but legitimately targeted by a higher number of logons (*e.g.*, servers), we weigh the sparseness of incoming logons by the ODS of the source.

Maximum-Sparseness-Factor (MSF) and Suspicious-User-Rate (SUR): A common characteristic of TAs is that they have been occasionally logged into with malicious intent. The MSF of a particular host denotes the ODS of the source that is most likely to be malicious and that has sparsely logged into that host. The higher is the MSF of a host the more likely it is a TA. The SUR of a given host is the proportion of usernames used to sparsely log into the host.

Attack Score (AS): The AS of a host reflects the likelihood of it being a TA. The higher the AS of a host, the more like it is a TA. AS is the product of MSF and SUR, hence it is correlated with MSF and SUR. Experiments with AS show a promising boost with respect to precision and recall. As the product of MSF and SUR, AS serves as an evident sign that the host has been tempted by an actively probing source host. It further reveals that selected ML models can not capture the product relationship of different features. With all the features, the classifier can better distinguish the boundary between TAs and benign hosts.

Further details on each and every extracted feature is available in Appendix A.

Algorithm 1 Sparseness function (SF)

```
input  : Source host $Src$, username $Usr$, destination host $Dst$, thresholds $\theta$, $\beta$
output: $Sparseness$, a sparseness score of event defined by $Src$, $Usr$, and $Dst$
1: Initialize $Events$ to all events in authentication log
2: $Sparseness \leftarrow 0$
   /* filter(*) is a no-op, $countByDays()$ counts the numbers of days where the events
   occur */
3: $TotalDays \leftarrow Events.filter(Src, Usr, Dst).countByDays()$
4: if $TotalDays \leq \theta$ then
5:     $Sparseness \leftarrow max(TotalDays * \beta - Events.count(), 0)$
6: end if
7: return $Sparseness$
```

3.3 ML Techniques

With graph-based features extracted, we evaluate several ML techniques to detect TAs during LM. We start with decision tree (DT), a non-parametric supervised learning method. We also leverage random forest (RF), which is a classifier that uses multiple DTs to improve classification performance and avoid over-fitting. LogitBoost (LB) is another learning algorithm based on DT that we leverage in our evaluation. We also assess logistic regression (LR), which is very efficient and does not require feature scaling. However, its performance deteriorates with highly correlated features. We evaluate the above ML algorithms along with other well known algorithms, such as support vector machine, k-NN, and gaussian naïve bayes. However, we do not discuss the latter as they do not perform well in our evaluation.

3.4 Evaluation Metrics

In order to measure the performance of ML models, we use a variety of metrics. These include:

$$Precision = \frac{\text{True Positive}}{\text{True Positive + False Positive}} \times 100$$

$$Recall = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \times 100$$

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

The precision and recall are better criterion compared to accuracy, false positive rate, and true negative rate, to assess the performance of a classifier when the dataset is imbalanced. The F1 score is essentially a harmonic mean of precision and recall, which represents the overall performance of a classifier. A higher F1 score indicates both low false positives and low false negatives (*i.e.*, true TAs are identified without raising many false alarms). In addition, we plot receiver operating characteristic (ROC) curve to illustrate the performance of a classifier at different classification thresholds. We also calculate AUC to quantify ML performance.

Chapter 4

Evaluation

4.1 Environment

4.1.1 Hardware

We perform data analysis and pre-processing on a cluster of four nodes, each of which has a Intel(R) Xeon(R) 3.30GHZ CPU and 16GB RAM. These nodes are interconnected using 10Gbps Ethernet. ML model training, validation and testing is performed on a machine equipped with 2 x Intel(R) Xeon(R) 2.20GHz CPU and 384 GB RAM.

4.1.2 Software

We leverage Numpy [43], Scipy [29], and Pandas [40] for data pre-processing. Imbalanced-learn [36] is employed for balancing the training datasets, while Scikit-learn [46] is used for building ML models.

4.2 Results

4.2.1 Feature Selection

We start by evaluating the performance of different ML classifiers with graph-based features extracted from authentication logs and network flow logs. Table 4.1 showcases the result of k-fold cross-validation ($k = 10$) using a total of 35 features (6 flow-based and 29 authentication-based features) extracted from the first 30 days of the LANL dataset. We choose θ and β based on trial-and-error and the frequency of benign activities in the dataset. Most authentication events for a given combination of (src, $username$, dst) occur for more than three times per day and exist over three days. Hence, we set $\theta = \beta = 3$. The parameters for the ML techniques are set based on their performance *i.e.*, we choose the parameters that exhibit the best result in TA detection. DT is set to a maximum depth of 6, while RF uses 400 as the number of estimator with a maximum depth of 12. LB uses 100 estimators and a DT regressor with a maximum depth of 3. LR uses a tolerance of 0.0001 and a regularization strength of 1.

Table 4.1: ML performance using flow- and authentication-based features (35 features)

ML model	Precision	Recall	F1 score	Training time (s)
DT	75.62%	75.15%	0.75	0.14
RF	79.99%	79.27%	0.79	3.31
LB	80.31%	80.29%	0.80	6.69
LR	31.10%	5.47%	0.09	4.04

With the exception of LR, which performs poorly, the other ML techniques classify TAs with relatively high precision and recall (over 75%). LB outperforms DT and RF with the highest F1 score. The number of ML features not only influence the computational overhead, but can also result in model over-fitting. Hence, in order to reduce the number of features and identify the ideal feature set for TA detection, we then restrict the feature set to authentication-based features. As depicted in Table 4.2, with the exception of RL whose performance increases significantly, we witness a marginal performance degradation when discarding flow-based features. RF outperforms other classifiers, while saving about 14s in feature extraction time. The lackluster performance of the flow-based features can be attributed to the inferior quality of flow data in the LANL dataset, as discussed in Chapter 3. This undermines the suitability of flow-based features to detect TAs during LM in this particular dataset.

Next, we study the correlation between authentication-based features to further reduce the features for TA detection. Table 4.3 shows that among the 29 features, 11 (column features) are correlated with 4 others (row features), with a Pearson coefficient exceeding 0.6. Hence, we remove all the 11 correlated features from the feature set. Many of these features report on the daily average logon times; per host, per user, and per (host, user)

Table 4.2: ML performance using authentication-based feature set (29 features)

ML model	Precision	Recall	F1 score	Training time (s)
DT	75.26%	75.77%	0.75	0.11
RF	81.36%	80.12%	0.81	3.11
LB	79.64%	79.76%	0.79	5.46
LR	61.31%	53.56%	0.52	6.13

Table 4.3: Pearson correlation matrix for most correlated authentication-based features

FID	2	3	10	11	15	18	22	23	24	26	27
16	0.9	0.9	0.9	0.9	0.9	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.9	0.0	0.8
25	0.0	0.0	0.0	0.0	0.0	0.4	0.6	0.0	0.0	0.1	0.0
29	0.0	0.0	0.0	0.0	0.0	0.6	0.4	0.0	0.0	0.6	0.0

combination. These features are too generic and fail to describe the true nature of LM, which makes them less discriminative in tagging TAs. We further remove the *out-degree-avg-frequency* feature. This latter feature is only significant if there is evidence that the TA is compromised and is actively attempting to move laterally, which is not the case in this dataset. We re-evaluate the ML techniques after removing the 12 aforementioned features. Appendix A provides the Feature IDs (FIDs) for the authentication-based features.

As depicted in Table 4.4, the F1 score for all ML techniques improve with RF outperforming all other classifiers. Furthermore, LR shows the highest improvement in TA detection with an F1 score increase of 8%. Even though DT and LB are immune to highly correlated features [53], we notice a slight increase in their performance. The removed features primarily pertain to standard deviation and out-degree. On a single host, different users can have distinct authentication patterns, which will result in high values for standard deviation-based features, causing confusion for the classifiers. Furthermore, TAs do not necessarily have an exploring behaviour, thus out-degree-based features can also degrade the classifier performance. Hence, in the following experiments we use the reduced feature set of 17 authentication-based features.

Table 4.4: ML performance on reduced authentication-based feature set (17 features)

ML model	Precision	Recall	F1 score	Training time (s)
DT	77.59%	75.45%	0.76	0.05
RF	83.72%	81.23%	0.82	2.06
LB	78.99%	80.25%	0.80	2.26
LR	68.13%	54.55%	0.60	5.99

4.2.2 Ensemble Learning

In an effort to improve the performance of the stand-alone ML models for TA detection, we consolidate them using ensemble learning. Due to the lackluster performance of LR in comparison to other ML models (*cf.*, Table 4.4), we remove it from the list of potential classifiers in the ensemble approach. First, we employ the majority voting (MV) algorithm [35] that leverages all ML models in the ensemble in a uniform manner, and use k-fold cross-validation ($k = 10$) on the first 30 days of the LANL dataset. The detection of TAs during LM using MV over RF, LB and DT is shown in Table 4.5. However, this results in an inferior performance to stand-alone RF, since low performing classifiers can influence the voting process.

Table 4.5: Ensemble learning using majority voting

Ensemble	Precision	Recall	F1 score	Training time (s)
RF, LB, DT	80%	80.67%	0.80	2.60

Evidently, MV is unable to boost the performance of the best stand-alone classifier. Therefore, we explore another ensemble approach, namely weighted voting (WV) [23], where we can assign weights to ML models based on their stand-alone performance. Intuitively, this can identify a higher number of true positives (*i.e.*, TAs during LM) that are missed by RF, the best performing stand-alone classifier. However, with multiple combination of weights assigned to the ML models in Table 4.6, stand-alone RF still outperforms WV. This reveals that at the classification boundary where RF is unable to differentiate between benign and malicious points, LB and DT also suffer and do not facilitate better performance. Besides, these ensemble approaches increase training time, undermining their suitability for early LM detection. Therefore, we choose the stand-alone RF classifier as *our model* for further experiments.

Table 4.6: Ensemble learning using weighted voting, prioritizing stand-alone ML performance

RF	LB	DT	Precision	Recall	F1 score	Training time (s)
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	80.00%	80.41%	0.80	2.86
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	80.70%	80.89%	0.81	2.78
$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{6}$	80.68%	80.97%	0.81	2.60

4.2.3 Balancing the Dataset

We further evaluate the robustness of our ML model by training and testing it on logs recorded on different days. Redteam activities are only conducted on certain days, generating malicious events that account for a very small fraction of the total number of authentication events (*i.e.*, less than 0.0001%). Therefore, we reserve day 9, the day with the highest number of malicious authentication events, for testing, while the remaining days are chosen for training the model. We evaluate several well known sampling algorithms (*cf.*, Chapter 3) to balance the training dataset. For each sampling algorithm, we use distinct seeds across 5 iterations and compute the average for each metric. These seeds are consistent across the sampling algorithms. Furthermore, each sampling algorithm has its own best sampling rate *i.e.*, the ratio of TA versus Benign (TA/Benign). Hence, we experiment with different sampling rates and select the best sampling rate to portray the corresponding results.

Over-sampling The comparison of three different over-sampling algorithms, namely ROS, SMOTE, and ADASYN, is highlighted in Table 4.7. SMOTE results in the second highest recall, as synthesizing minority points help in stressing the TA class. However, the randomness in synthetic points do not capture the true nature of original TAs, resulting in a lower precision. In contrast, ADASYN achieves better precision and recall. It generates synthetic points closer to the decision boundary, thus enabling the classifier to better distinguish TAs from benign hosts. As opposed to over-sampling only a portion of the minority points, ROS simply replicates TAs, which stresses on all TAs. Uniformly stressing on all TAs preserve the originality of TA class and its behavior to a large extent in comparison to synthesizing, thus resulting in the highest precision. Each algorithm over-samples the dataset with the same sampling rate, thus the training time (TT) is similar. In contrast to ADASYN, ROS and SMOTE consume less sampling time (ST) due to their simpler sampling mechanism.

Table 4.7: Over-sampling with different algorithms
(17 features)

Algorithm	TA/Benign	Precision	Recall	F1 score	ST (s)	TT (s)
ROS	0.02	62.34%	95.36%	0.7539	0.01	4.27
SMOTE	0.02	61.99%	95.71%	0.7524	0.01	4.62
ADASYN	0.02	62.08%	96.07%	0.7542	0.06	4.66

Under-sampling Recall that RUS randomly removes samples from the majority class. This may result in a high number of benign (majority class) hosts that have similar traits as certain class of TAs, negatively impacting precision. This is evident in the lower precision of RUS in comparison to ConNN, as shown in Table 4.8. But the TAs that starkly differ from the benign hosts are still classified with high recall. A similar affect can be seen with RENN, which removes benign hosts that are not very similar to their neighbors. In contrast, ConNN preserves the benign hosts that are different from their neighbors. Therefore, under-sampling with ConNN results in the best F1 score with precision and recall of 95.12% and 62.47%, respectively. Due to its simplicity, RUS incurs the least sampling time. In contrast, ConNN suffers from the highest sampling time, but it also reduces the number of benign hosts to the largest extent, which positively impacts the training time.

Table 4.8: Under-sampling with different algorithms
(17 features)

Algorithm	TA/Benign	Precision	Recall	F1 score	ST (s)	TT (s)
RUS	0.02	60.9%	96.55%	0.747	0.01	2.45
ConNN	0.51	62.47%	95.12%	0.754	130.88	0.99
RENN	0.01	60.07%	97.62%	0.743	2.81	3.32

Comparison We highlight the over- and under-sampling algorithms with the highest F1 score in Table 4.9, along with no sampling (*i.e.*, unbalanced training dataset). As evident, ADASYN increases the precision and recall by 0.54% and 0.84%, respectively. However, this comes at the cost of increased sampling and training times, which undermines its suitability. On the other hand, ConNN increases precision by 0.93%. However, its sampling time is very high in comparison to training without any sampling. Thus, we proceed without any sampling to detect TAs during LM in the LANL dataset.

Table 4.9: Comparing different algorithms
(17 features)

Algorithm	TA/Benign	Precision	Recall	F1 score	ST (s)	TT (s)
ADASYN	0.02	62.08%	96.07%	0.7542	0.06	4.66
ConNN	0.51	62.47%	95.12%	0.7541	130.88	0.99
Unbalanced	0.01	61.54%	95.23%	0.7476	0	3.61

4.2.4 Comparative Analysis

To further evaluate our approach, we compare our model with two state-of-the-art approaches for LM detection. We implement the approaches in Chen *et al.* [15] and Kaiafas *et al.* [31]. To achieve a fair comparison, we balance the dataset according to the algorithm in [15], which preserves the redteam events while under-sampling the benign activities. Due to scalability issues in [31], we only leverage data for k-fold cross-validation ($k = 10$) from day 9. As depicted in Table 4.10, our model outperforms Chen *et al.* in precision, recall and F1 score. However, our approach consumes more feature extraction time (FET) and model training time.

Kaiafas *et al.* marginally outperforms our model in recall, with an improvement of 0.02 in F1 score. However, their feature extraction and model training times are magnitudes higher than both Chen *et al.* and our approach. In the balanced dataset, there are about 97,000 authentication events and their overhead is largely due to feature extraction for each individual event. In contrast, our approach strikes a balance between performance and overhead.

Table 4.10: TA detection using stand-alone RF and
cross-validation versus ([15], [31])

Classifier	Precision	Recall	F1 score	FET (s)	TT (s)
Our Model	97.02%	93.04%	0.95	169.35	0.145
Chen *et al.*	7.24%	73.12%	0.13	0.69	0.529
Kaiafas *et al.*	100%	93.47%	0.97	100.81	2333.24

Followed by cross-validation, we evaluate the robustness of the aforementioned approaches on never seen data. Thus, we leverage authentication events from day 9 as the test dataset, while the remainder of the dataset (*i.e.*, 29 days) is used for training. In this case, the training dataset is composed of over 220 million log entries, which can potentially introduce a lot of noise. As depicted in Table 4.11, the model from Chen *et al.* fails miserably with a near-zero recall when tested on never seen data. The authors in [15] leverage features, including network traffic amount, sending packet amount, authentication amount and DNS queries amount, *etc.* These generic statistical features fail to distinguish TAs in a noisy environment. Unfortunately, we are unable to extract features for Kaiafas *et al.* for this robustness evaluation in a reasonable amount of time. Thus, the robustness evaluation for their model is unavailable. In contrast, our model shows remarkable performance with a recall and F1 score of 98.81% and 0.75, respectively.

Table 4.11: Robustness of TA detection using stand-alone RF versus ([15], [31])

Classifier	Precision	Recall	F1 score	FET (s)	TT (s)
Our Model	60.58%	98.81%	0.75	2210.45	3.95
Chen *et al.*	2.98%	3.16%	0.030	823.51	59.45
Kaiafas *et al.*	—	—	—	> 360000	—

Nevertheless, to compare the robustness of Kaiafas *et al.* we reduce the cardinality of the training dataset from the previous experiment. Data from days 13, 14, and 15 is used for training, while day 9 is reserved for testing. For a fair comparison, we leverage the under-sampling method from Chen *et al.* for both comparative models. Note that Kaiafas *et al.* do not expose their sampling approach in detail. Furthermore, no sampling is applied to our model. As shown in Table 4.12, our model significantly outperforms other approaches. Even though Kaiafas *et al.* perform quite well in cross-validation, they fail in robustness to never seen TAs.

The authors in [31] extract features, including frequency, first occurrence tag, diversity of user, *etc.* However, these features fail to differentiate the TAs from the benign hosts for large networks. Due to the diversity of different authentication events, they can result in hosts having similar values with regard to less crafted features, such as the number of successful/failed authentication events. Such noise will influence the performance of ML models that leverage less-thought-of features. However, with features based on the degree of sparse events, our model is able to filter out noise and differentiate TAs. Fig. 4.1 shows the ROC curve, which indicates that our model has the highest AUC of 0.995. Note that Kaiafas's model is using MV, which is not feasible for plotting as a ROC curve. In comparison to previous robustness result, our model shows a marginal loss in precision and recall. However, our model out classes other approaches, with a high precision of over 94%, while the F1 score is the highest at 0.74.

Table 4.12: Robustness of TA detection using stand-alone RF versus ([15], [31]) on a reduced training dataset

Classifier	Precision	Recall	F1 score	FET (s)	TT (s)
Our Model	61.24%	94.05%	0.74	475.46	2.56
Chen *et al.*	4.64%	9.52%	0.06	11.22	0.66
Kaiafas *et al.*	9.58%	45.83%	0.16	40488.56	1903.24

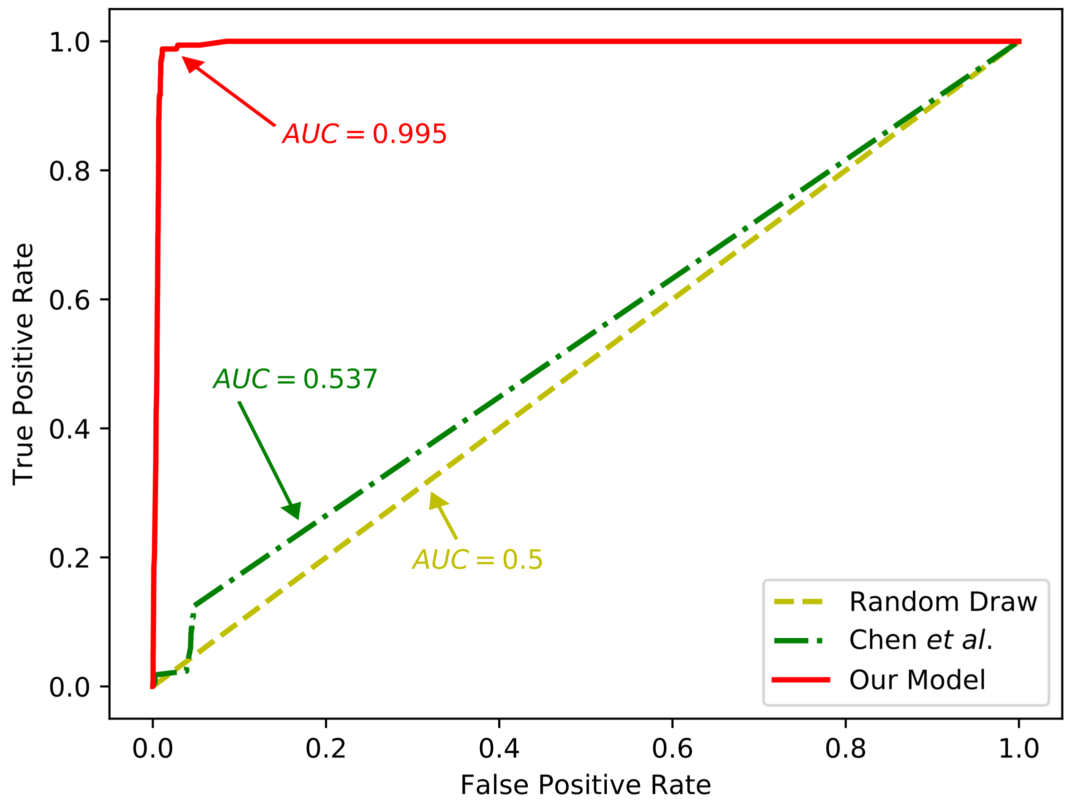

Figure 4.1: ROC for robustness in TA detection using stand-alone RF vs. ([15], [31]), with days 13, 14 and 15 for training, and day 9 reserved for testing

Chapter 5

Adversary Study

In APT, attackers during LM strive to introduce minimal influence. Highly skilled attackers can trick a system by adding small randomness in their activities or imitating benign behavior after long-term scouting. However, in practice, it is very difficult for an attacker to collect behaviour information of benign users. Therefore, we study the influence of adversarial attempts for the former scenario. We assume that the attacker cannot access behaviour profile of benign users, the model or the training data. Therefore, the adversary introduces variations in attack patterns to potentially trick the classifier. Since our approach relies on authentication patterns, variations in other aspects such as system vulnerability and perturbing tools, will not influence the classifier's performance. Our approach capitalizes on the infrequency of malicious authentication events. Thus, we evaluate the influence of an adversary perturbing its probing pattern. More specifically, this includes perturbing the number of TAs and the frequency of malicious authentication events, as shown in Fig. 5.1. An attacker can both increase the attacking frequency (in b) and decrease it. Similarly, an attacker can also increase the number of TAs (in c) or reduce it. We start by exploring perturbations in both of these aspects in Sections 5.1 and 5.2.

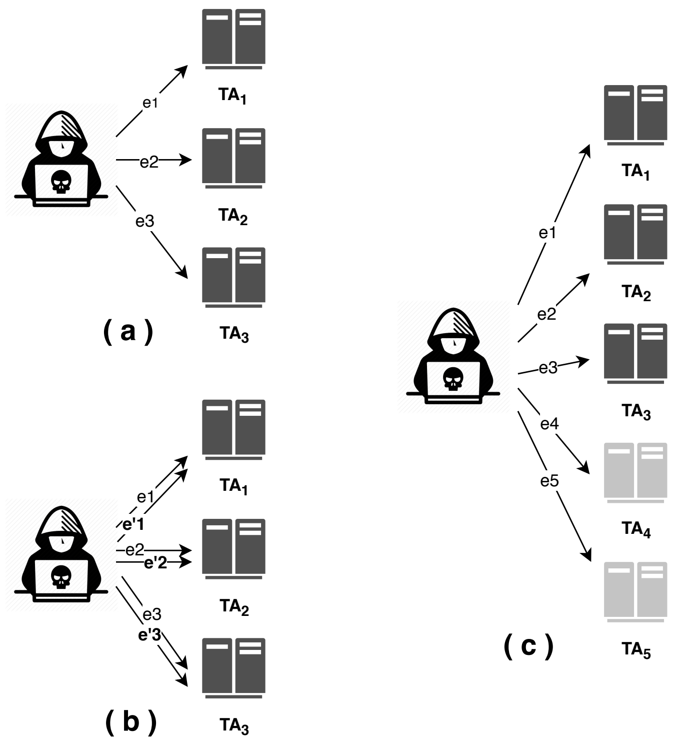

Figure 5.1: Attacker changes probing pattern: (a) the original attack, (b) increasing attack frequency (c) increasing the attacking scale (*i.e.*, attack more TAs). Note that arrow labels correspond to authentication events.

5.1 Perturbing Number of TAs

To simulate attackers probing different number of TAs, we first define the *Ratio* between the sampled TAs and the original TAs as follows:

$$Ratio = \frac{\text{Number of TAs after sampling}}{\text{Number of TAs in the original dataset}}$$

For *Ratio* greater than 1 (*i.e.*, increase in number of TAs), we randomly select $(Ratio - 1) \times Original\ Number\ of\ TAs$ from within benign hosts. Then for each of the chosen new host (*i.e.*, victim), we simulate the attack from the original redteam in a one-to-one manner. An example is illustrated in Fig. 5.2. Similarly, for *Ratio* smaller than 1 (*i.e.*, decrease in number of TAs), we randomly select $Ratio \times Original\ Number\ of\ TAs$ from the redteam, and only preserve redteam events related to the selected TAs.

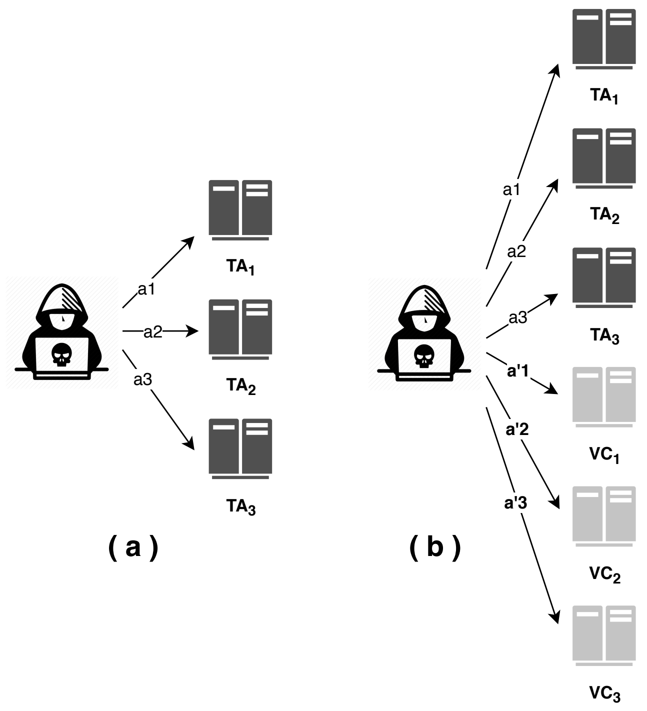

Figure 5.2: Example of increase in number of TAs with $Ratio = 2$: (a) original attack, (b) attacking more TAs, for each new host (*i.e.*, victim VC_i for corresponding original TA_i) map to it the original attack a_i on TA_i *i.e.*, a'_i. Note, attack a_i on TA_i is a set of authentication events.

Decreasing the $Ratio$ can reduce the sparse related graph-based features. When attacking fewer TAs, the ODS of the attacking (source) host will decrease. Consequentially, the $WIDS_{(usr,src)}(dst_j)$ feature will also decrease, as it is weighted by the source ODS. We assert this claim by inspecting the features for each host in our experiments. Fig. 5.3 shows the AUC with respect to different $Ratio$. At a high level, the decrease in $Ratio$ does not severely impact the true positive rate (TPR) and the false positive rate (FPR). However, upon zooming on the upper left corner (*i.e.*, the balance between TPR and FPR), as shown in Fig. 5.4, there is a noticeable variation in TPR at the balance point. We further highlight the corresponding results in Table 5.1.

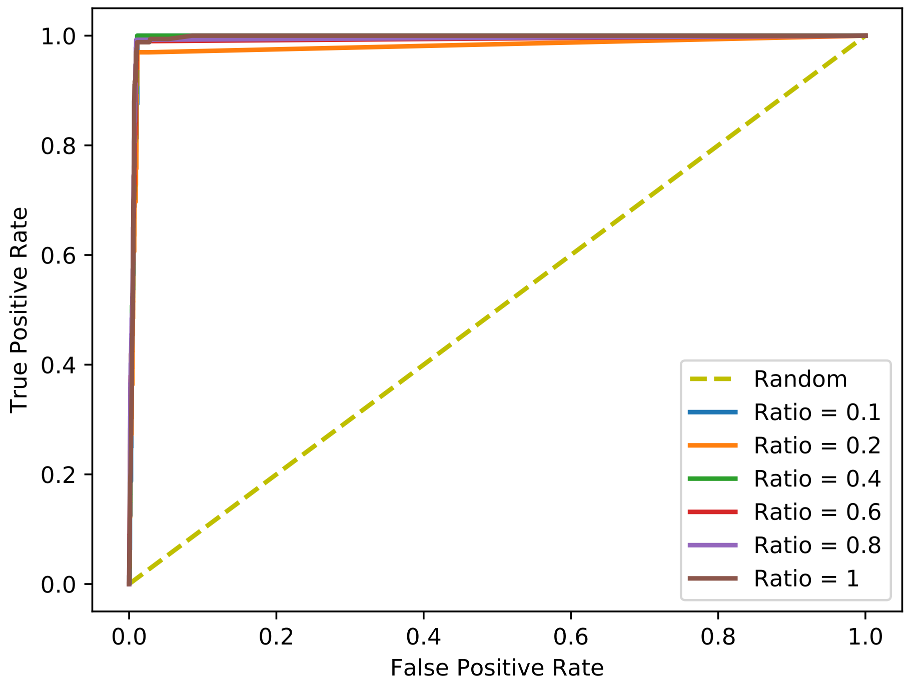

Figure 5.3: AUC with decrease in number of TAs and *Ratio*

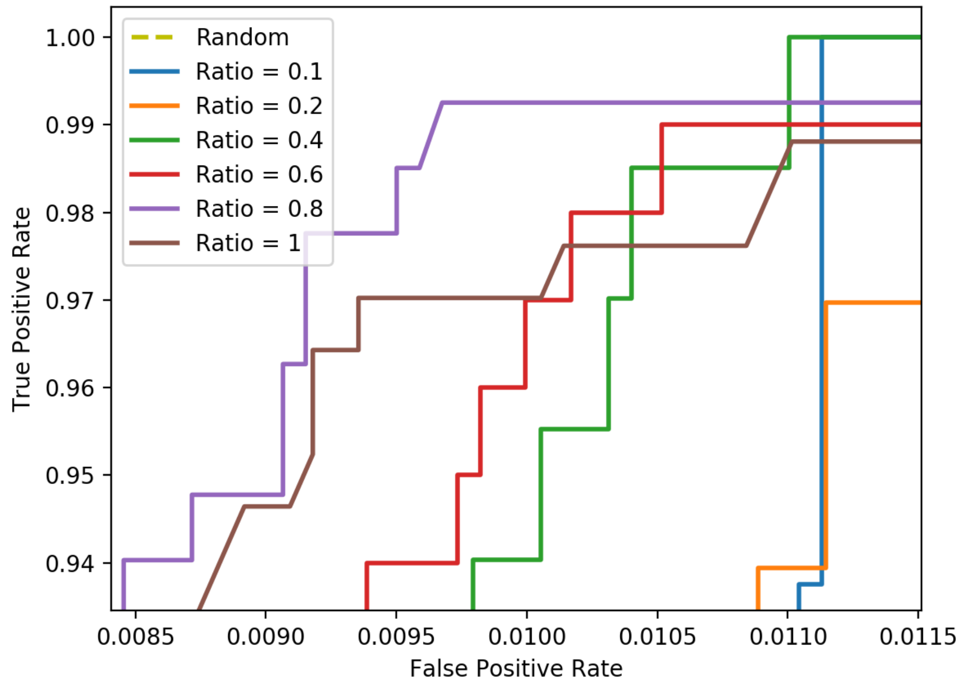

Figure 5.4: AUC with decrease in number of TAs and *Ratio* (zoom view)

Table 5.1: Decrease in number of TAs

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	60.58%	98.81%	0.751	166	108	2
0.8	53.85%	99.25%	0.6982	133	114	1
0.6	45%	99%	0.6188	99	121	1
0.4	34.54%	100%	0.513	67	127	0
0.2	19.76%	100%	0.3299	33	134	0
0.1	10.46%	100%	0.189	16	137	0

Evidently, the number of FP increases as the *Ratio* decreases. By inspecting the data points after sampling, we observe that for most benign hosts the sparse related features stay at a very low level. For example, the average $WIDS_{(usr,src)}(dst_j)$ of TAs is around ten times that of benign hosts. Furthermore, we notice a drop in these features for TAs. Therefore, we concur that the classifier is not aggressively relying on these distinctive features. One way to adjust the sensitivity to sparse features is to tune the classifier threshold. Table 5.2 shows the result with the threshold values that lead to the best F1 score. There is a clear increase in precision when compared to the use of a uniform threshold. However, manually tuning the classifier threshold is impractical after deployment. Therefore, to directly stress on sparse related features, we further explore by only considering the corresponding features.

Table 5.2: Decrease in number of TAs and classifier threshold according to F1 score

Ratio	Precision	Recall	F1 score	TP	FP	FN	Threshold
1	60.58%	98.81%	0.751	166	108	2	0.3
0.8	54.29%	99.25%	0.7024	133	112	1	0.32
0.6	45.37%	98%	0.620	98	118	2	0.33
0.4	34.74%	98.51%	0.5136	66	124	1	0.33
0.2	20.75%	100%	0.3438	33	126	0	0.35
0.1	11.03%	100%	0.1988	16	129	0	0.36

Table 5.3 shows the result using sparse related features only *i.e.*, without features such as $ID_{src}(dst_j)$, $ID_{usr}(dst_j)$, and $ID_{(usr,src)}(dst_j)$. Evidently, the classifier can not perform well at the decision boundary where benign hosts and TAs are very similar. Therefore, there is a significant increase in FP and FN. With these observation, it is clear that sparse related features can effectively identify TAs, but at the same time, misclassify benign hosts that have similar probing behavior as TAs. Furthermore, non-sparse related features serve as a support to rule out these FNs, thus it is not possible to ensure classifier robustness to smaller *Ratio* pertubations by leveraging only the sparse related features.

Table 5.3: Decrease in number of TAs and *Ratio*, using sparse related features only

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	54.05%	95.24%	0.690	160	136	8
0.8	47.41%	95.52%	0.634	128	142	6
0.6	39.34%	96%	0.558	96	148	4
0.4	30.14%	98.51%	0.4615	66	153	1
0.2	17.10%	100%	0.292	33	160	0
0.1	8.38%	93.75%	0.1538	15	164	1

We then investigate when attackers increase the number of TAs. There is no significant drop in TPR and FPR, asserting classifier robustness to this kind of perturbation, as illustrated in Fig. 5.5. Table 5.4 shows a significant increase in precision, while there is a decrease in recall. Since we choose more benign hosts as TAs and perform similar probing activities on them, the classifier is able to identify them with higher precision. However, it is counter intuitive for FN to increase significantly, since the classifier has seen similar activities. Therefore, we further investigate with only sparse features. Table 5.5 shows more stable results with respect to recall. Therefore, using only sparse related features achieves stable recall at the cost of lower precision.

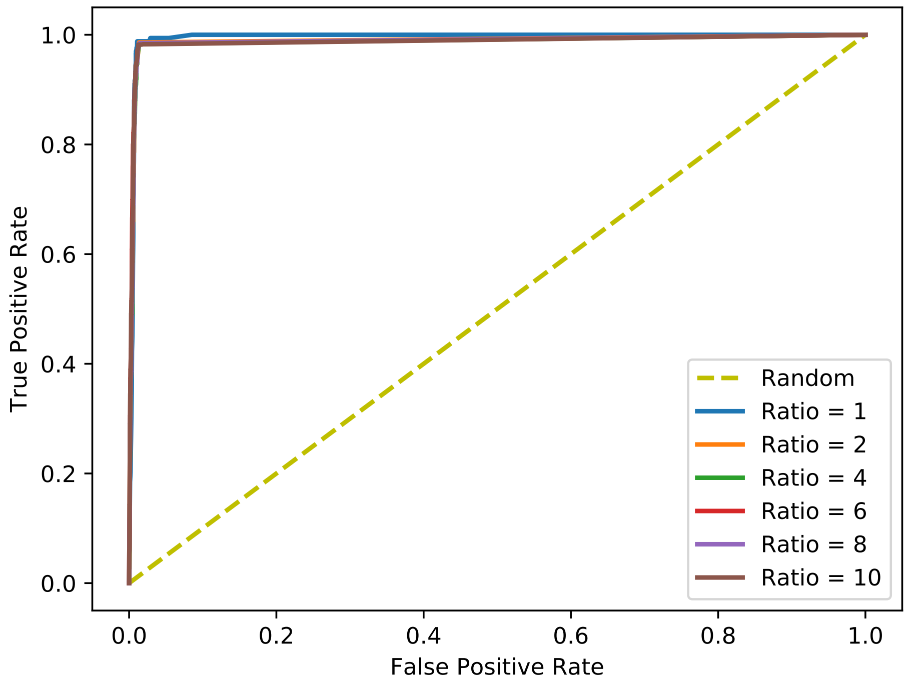

Figure 5.5: AUC with increase in number of TAs and *Ratio*

Table 5.4: Increase in number of TAs, using uniform threshold

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	60.58%	98.81%	0.751	166	108	2
2	74.76%	94.35%	0.834	317	107	19
4	86.43%	92.86%	0.895	624	98	48
6	90.68%	91.67%	0.912	924	95	84
8	93.37%	92.26%	0.928	1240	88	104
10	94.38%	91.96%	0.932	1545	92	135

Table 5.5: Increase in number of TAs, using sparse features only

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	54.05%	95.24%	0.690	160	136	8
2	70.55%	95.54%	0.812	321	134	15
4	83.99%	95.24%	0.893	640	122	32
6	88.96%	95.93%	0.923	967	120	41
8	91.86%	95.68%	0.937	1286	114	58
10	93.44%	95.83%	0.946	1610	113	70

5.2 Perturbing Attack Frequency

Another aspect that we explore is frequency with respect to an attacker's probing activity (PA). Here We define *Ratio* as follows:

$$Ratio = \frac{\text{Number of PAs after sampling}}{\text{Number of PAs in the original dataset}}$$

For increasing the attack frequency, we simulate additional attacks by replicating the original attacks on a TA by a factor of *Ratio*, as shown in Fig. 5.6. For each new attack we also add randomness to its attack time. Similarly, for a decrease in attack frequency, we randomly sample the attacks on each TA to the *Ratio* of the original attack.

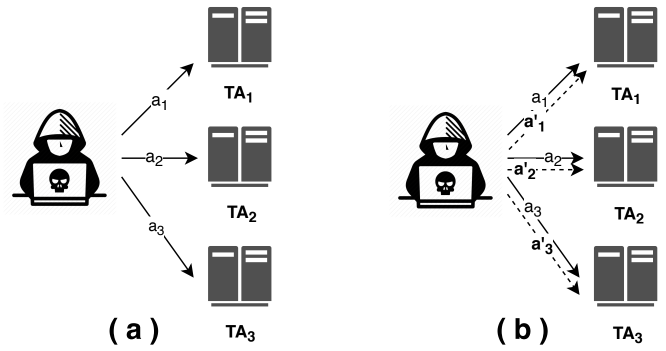

Figure 5.6: Example of increase in attack frequency with $Ratio = 2$: (a) the original attack, (b) for each attack a_i on the original TA_i, we add randomness to its attack time, which results in a'_i that is simulated on TA_i. Note, attack a_i on TA_i is a set of authentication events.

Fig. 5.7 shows the AUC of decreasing the attack frequency. Since the features are built atop sparse authentication events, reducing the frequency further emphasizes the sparse related features. Thus, there is no significant drop in TPR or FPR. Fig. 5.8 shows the upper left corner of the AUC, where the optimum balance of TPR and FPR resides. In general, the model performs better as the *Ratio* decreases. Table 5.6 shows the precision, recall, and F1 score with decreasing *Ratio*. As the *Ratio* decreases, all metrics increase steadily. That is, as attacks get less frequent, sparse related features become more pronounced, which helps the classifier better identify TAs. Hence, our model shows robustness to infrequent attacks.

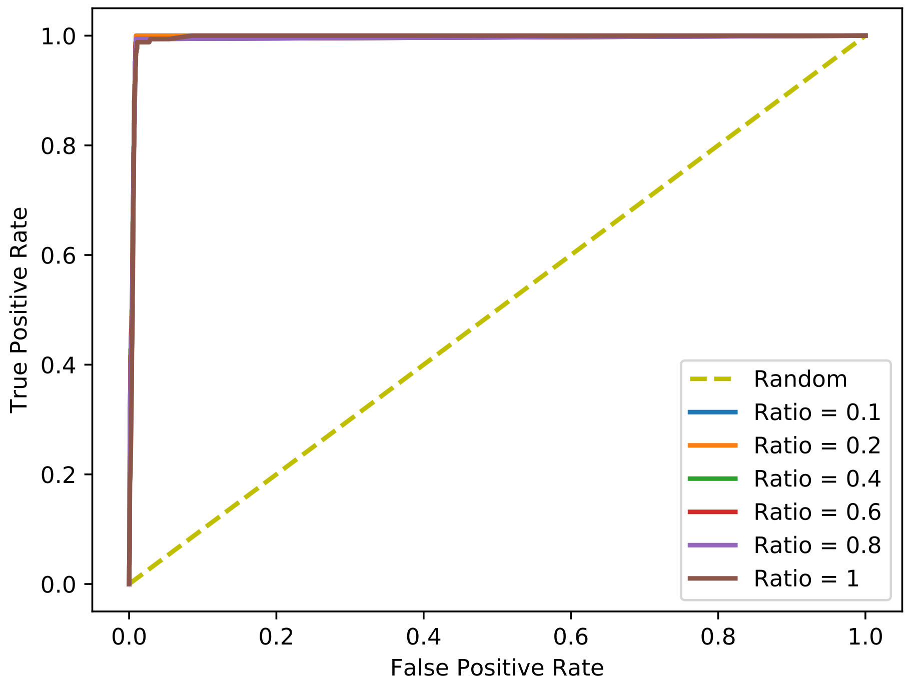

Figure 5.7: AUC with decrease in attack frequency

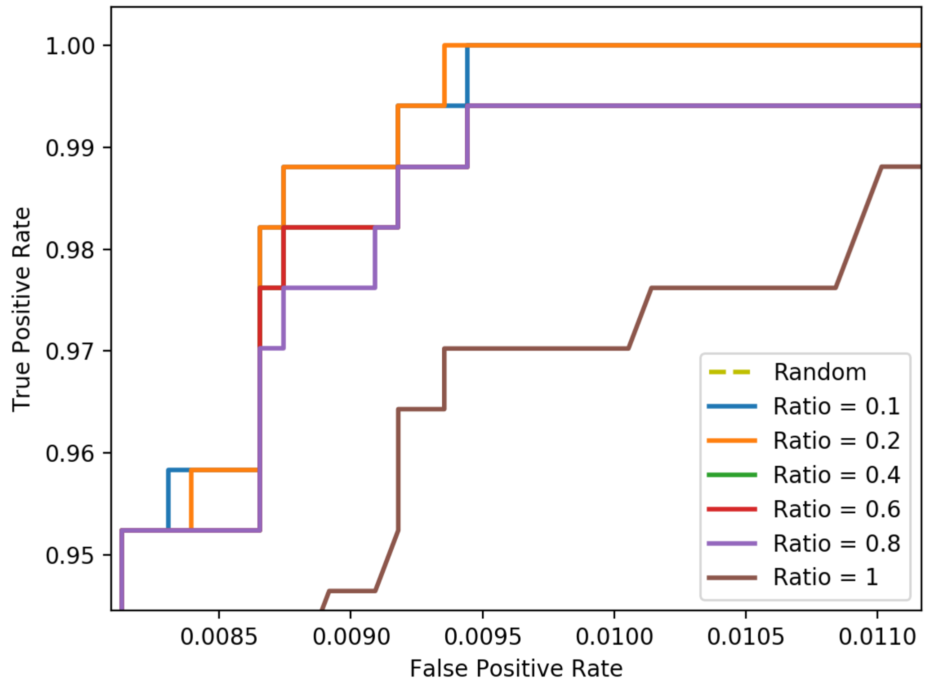

Figure 5.8: AUC with decrease in attack frequency (zoomed view)

Table 5.6: Decrease in malicious authentication events frequency with different *Ratio*

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	60.58%	98.81%	0.751	166	108	2
0.8	60.73%	99.40%	0.754	167	108	1
0.6	60.73%	99.40%	0.754	167	108	1
0.4	60.73%	99.40%	0.754	167	108	1
0.2	60.87%	100%	0.757	168	108	0
0.1	60.87%	100%	0.757	168	108	0

Table 5.7: Decrease in malicious authentication events frequency with different *Ratio*, using sparse features only

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	54.05%	95.24%	0.690	160	136	8
0.8	54%	96.43%	0.692	162	138	6
0.6	53.85%	95.83%	0.690	161	138	7
0.4	54%	96.43%	0.692	162	138	6
0.2	53.97%	97.02%	0.694	163	139	5
0.1	54.13%	97.62%	0.696	164	139	4

Figure 5.9 shows the AUC of increasing the attack frequency. As the *Ratio* increases, probing behavior becomes similar to regular benign activities, which diminishes the discriminating nature of sparse related features. As a result, both the precision and recall suffer, as shown in Tables 5.8 and 5.9. Even though employing all features show consistently better result, the less prominent sparse related features sabotage model performance.

In summary, our model shows robustness to attackers that probe more TAs or reduce the attack frequency. When the attack frequency increases or when fewer TAs are targeted, both precision and recall drop significantly. This is consistent with how the sparse features are influenced, more prominent sparse related graph-based features improve the classifier performance. The classifier itself is not extremely sensitive to sparse feature change, since other non-sparse related features can boost the precision. Tuning the threshold helps improve the recall by adjusting the classifier's sensitivity to sparse feature's increase. However, manually tuning the classifier threshold is not feasible in practice. Furthermore, using only sparse related features result in higher FPs and reduces the classifier's ability to differentiate TAs and benign hosts that are similar in the feature space. Therefore, we propose to keep all the features and solve the issue by perturbing the training data.

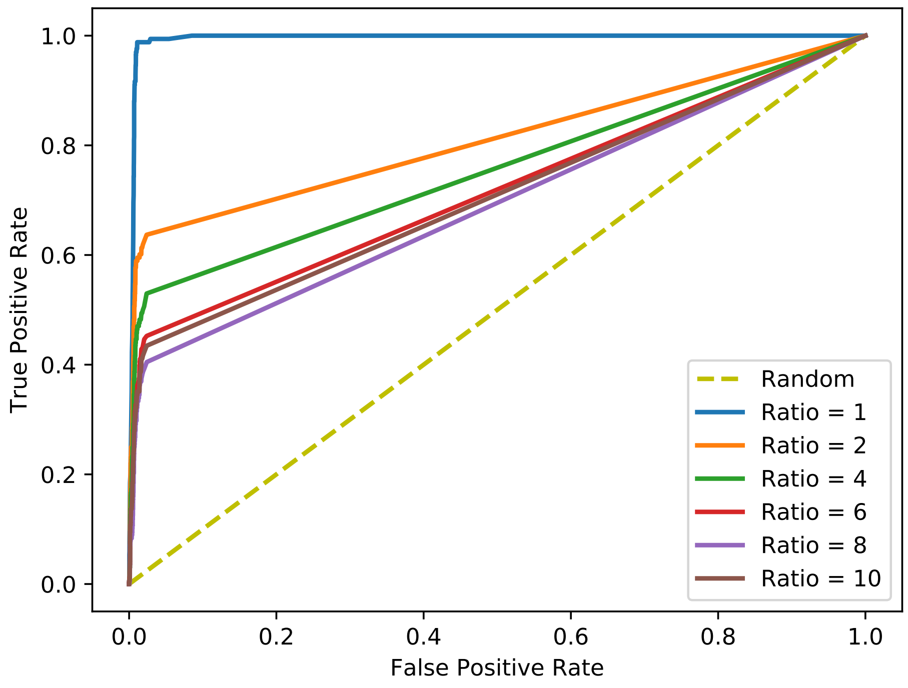

Figure 5.9: AUC with increase in attack frequency

Table 5.8: Increase in malicious authentication events frequency with different *Ratio*

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	60.58%	98.81%	0.751	166	108	2
2	47.83%	58.93%	0.528	99	108	69
4	42.17%	41.67%	0.419	70	96	98
6	36.43%	30.36%	0.331	51	89	117
8	33.33%	26.19%	0.293	44	88	124
10	21.74%	14.89%	0.177	25	90	143

Table 5.9: Increase in malicious authentication events frequency with different *Ratio*, using sparse features only

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	54.05%	95.24%	0.690	160	136	8
2	40.52%	55.95%	0.47	94	138	74
4	34.33%	41.07%	0.374	69	132	99
6	27.87%	30.36%	0.291	51	132	117
8	25.99%	27.38%	0.267	46	131	122
10	16.13%	14.88%	0.155	25	130	143

5.3 Adversarial Learning

With the above discussion, we conclude that our model is robust when the attacker decreases the attack frequency or increases the number of TAs. But when the attacker changes the attacking pattern in the other direction (*i.e.*, increases the attack frequency or decreases the number of TAs), our model is vulnerable. That is, we observe a drop in both precision and recall.

To alleviate this impact, we add synthetic TAs with perturbed attack patterns to enhance the classifier's ability in differentiating TAs from benign hosts in an adversarial setting. More specifically, we generate synthetic TAs by decreasing the number of TAs. As shown in Fig. 5.10, with a $Ratio = 0.6$, we randomly choose 60% of the original TAs, add randomness to their attack time (*i.e.*, for corresponding authentication events) and generate synthetic TAs (*i.e.*, TA's). Then we add the synthetic TAs to the training dataset, retrain our model and reevaluate its performance. After experimenting with different *Ratio*, we get the best result with $Ratio = 0.6$. Table 5.10 depicts the classifier performance after adding these synthetic TAs to the training dataset.

Comparing the results with that of the original experiment (*cf.*, Table 5.3), there are more FPs, which means lower precision, while the recall is restored to 100%. Adding the synthetic TAs in the training dataset makes the classifier more aggressive at the boundary. This results in a higher number of FPs, while enabling a 100% recall, our primary objective. Table 5.11 shows a similar influence on performance when the adversary increases the attack frequency (when compared with Table 5.8). Interestingly, the synthetic TAs generated by decreasing the number of TAs alleviates both vulnerabilities. Recall that perturbing attack behavior makes the sparse related features less prominent, causing the classifier to

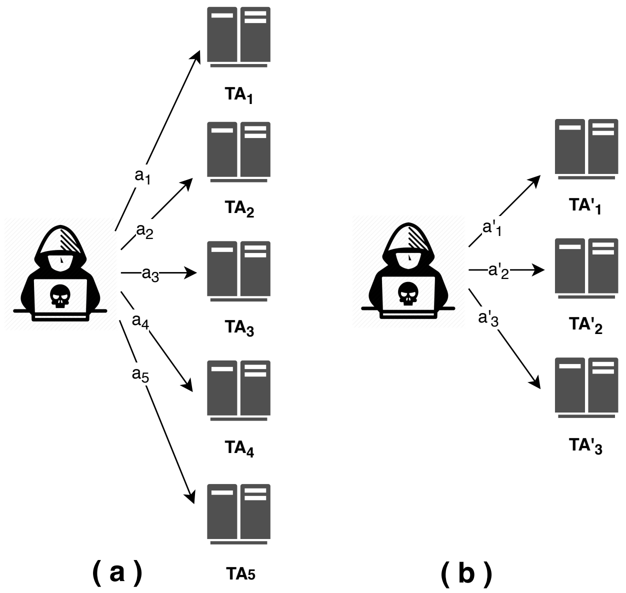

Figure 5.10: Example of generating synthetic TAs by decreasing the number of TAs with $Ratio = 0.6$: (a) the original attack, (b) randomly choose TA'_i from the original TA_i. Add randomness to each attack a_i's attack time, which results in a'_i that is simulated on TA'_i. Note, attack a_i on TA_i is a set of authentication events.

misclassify TAs with low sparse features. Adding synthetic TAs enables the classifier to correctly classify TAs with less prominent sparse features, making it robust to adversarial attempts.

Table 5.10: Decrease in number of TAs, adding synthetic TAs to training dataset ($Ratio = 0.6$)

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	53.53%	99.40%	0.6958	167	145	1
0.8	46.85%	100%	0.638	134	152	0
0.6	38.61%	100%	0.557	100	159	0
0.4	28.88%	100%	0.448	67	165	0
0.2	16.10%	100%	0.277	33	172	0
0.1	8.38%	100%	0.155	16	175	0

Table 5.11: Increase in malicious authentication events frequency, adding synthetic TAs to training dataset ($Ratio = 0.6$)

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	53.53%	99.40%	0.6958	167	145	1
2	46.51%	59.92%	0.522	100	115	68
4	41.08%	45.24%	0.431	76	109	92
6	33.74%	32.74%	0.332	55	108	113
8	31.41%	29.17%	0.302	49	107	119
10	21.43%	17.86%	0.195	30	110	138

Table 5.12: Decrease in number of TAs, adding synthetic TAs to training dataset ($Ratio = 0.1$)

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	53.55%	98.81%	0.6946	166	144	2
0.8	47.00%	99.25%	0.638	133	150	1
0.6	38.67%	99%	0.556	99	157	1
0.4	29.13%	100%	0.451	67	163	0
0.2	16.26%	100%	0.280	33	170	0
0.1	8.47%	100%	0.156	16	173	0

Table 5.13: Increase in malicious authentication events frequency, adding synthetic TAs to training dataset ($Ratio = 0.1$)

Ratio	Precision	Recall	F1 score	TP	FP	FN
1	53.55%	98.81%	0.6946	166	144	2
2	46.70%	58.93%	0.521	99	113	69
4	42.13%	44.64%	0.434	75	103	93
6	33.77%	30.95%	0.323	52	102	116
8	31.54%	27.98%	0.297	47	102	121
10	22.73%	17.86%	0.2	30	102	138

We expect this influence to decrease as the number of synthetic TAs added to the training dataset decreases. Table 5.12 asserts our conjecture. FP increases with a reduced number of TAs and so does FN. Therefore, we add synthetic TAs with $Ratio = 0.6$ in the training dataset, to achieve the best recall at the cost of slightly lower precision. In the two vulnerable cases of attackers increasing the attack frequency and probing smaller number of TAs, our approach results in an effective mitigation of adversarial attempts against our model. However, we do notice in Table 5.13 the false negative is still very high. We have explored adding synthetic TAs generated with reduced Ratio. By adjusting the number of TAs to add in the training dataset, we can potentially further improve the performance. Besides, adding synthetic TAs generated by increasing the frequency may also further mitigate the impact of the perturbation.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we propose a novel approach for detecting TAs during the LM phase of an APT attack. We explore graph-based features extracted from multiple data sources (*i.e.*, network flows and host authentication logs) in the LANL dataset. Among all the baseline features, we filter less impactful and correlated features to select the ideal feature set for TA detection and reduce computational overhead. To cope with the highly imbalanced nature of the dataset, different sampling algorithms are explored to improve classifier performance. The result shows that our approach is robust against imbalanced dataset. Our approach outperforms the other state-of-the-art approaches in TA detection on the LANL dataset.

However, our approach is limited by the poor quality of the LANL dataset. This prevents us from exploiting data from multiple sources for TA detection. This is largely due to the incompleteness of network traffic monitoring data. Apart from this, the sampling algorithms do not significantly boost the performance of classifiers, which needs further investigation. Furthermore, in situations where attackers evade detection by adding variation in their probing behaviour, we show that our model can be robust by adding synthetic TAs in the training data. As the data grows rapidly in an enterprise network, the employment of online learning would be valuable, both in terms of computation overhead and performance. This will also facilitate the adjustment of ML decision boundary after deployment.

6.2 Future Work

On top of our approach for TA detection during LM, several other avenues can be explored.

Evolving Sparse Function: Currently, the graph-based features are generated from a sparse function with thresholds. These parameters are not fully explored with different benign activity patterns. An adaptive sparse function that adjusts itself according to benign behavior can improve the robustness of TA detection.

Evolving Threshold: The threshold of the classifier reveals its sensitivity to the boundary data points. It has a big impact on precision and recall. The automatic evolving of the threshold has the potential of stabilizing the classifier's performance in different scenarios.

Hybrid Sampling: A simple sampling approach has been explored in this thesis. A more sophisticated sampling approach could further improve the performance.

References

[1] Basant Agarwal and Namita Mittal. Hybrid approach for detection of anomaly network traffic using data mining techniques. *Procedia Technology*, 6:996–1003, 2012.

[2] Shikha Agrawal and Jitendra Agrawal. Survey on anomaly detection using data mining techniques. *Procedia Computer Science*, 60:708–713, 2015.

[3] Hyrum S Anderson, Jonathan Woodbridge, and Bobby Filar. Deepdga: Adversarially-tuned domain generation and detection. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, pages 13–21. ACM, 2016.

[4] Giovanni Apruzzese and Michele Colajanni. Evading botnet detectors based on flows and random forest with adversarial samples. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[5] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo. Machine learning for cognitive network management. *IEEE Communications Magazine*, 56(1):158–165, 2018.

[6] Pierre Baldi. Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the International Conference on Unsupervised and Transfer Learning Workshop*, pages 37–50, 2011.

[7] Harinder Pal Singh Bhasin, Elizabeth Ramsdell, Albert Alva, Rajiv Sreedhar, and Medha Bhadkamkar. Data center application security: Lateral movement detection of malware using behavioral models. *SMU Data Science Review*, 1(2):10, 2018.

[8] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[9] A. Bohara, M. A. Noureddine, A. Fawaz, and W. H. Sanders. An unsupervised multi-detector approach for identifying malicious lateral movement. In *Proceedings of IEEE Symposium on Reliable Distributed Systems*, pages 224–233, 2017.

[10] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Springer Journal of Internet Services and Applications*, 9(1), 2018.

[11] Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the First Workshop on Machine Learning for Computing Systems*, pages 1–8, 2018.

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[13] Ashima Chawla, Brian Lee, Sheila Fallon, and Paul Jacob. Host based intrusion detection system with combined cnn/rnn model. In Carlos Alzate, Anna Monreale, Haytham Assem, Albert Bifet, Teodora Sandra Buda, Bora Caglayan, Brett Drury, Eva García-Martín, Ricard Gavaldà, Irena Koprinska, Stefan Kramer, Niklas Lavesson, Michael Madden, Ian Molloy, Maria-Irina Nicolae, and Mathieu Sinn, editors, *ECML PKDD 2018 Workshops*, pages 149–158, Cham, 2019. Springer International Publishing.

[14] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[15] M. Chen, Y. Yao, J. Liu, B. Jiang, L. Su, and Z. Lu. A novel approach for identifying lateral movement attacks based on network embedding. In *Proceedings of IEEE International Conf. on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications*, pages 708–715, 2018.

[16] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In *IFIP International Conference on Communications and Multimedia Security*, pages 63–72. Springer, 2014.

[17] Roshan Chitrakar and Chuanhe Huang. Anomaly based intrusion detection using hybrid learning approach of combining k-medoids clustering and naive bayes classification. In *Proceedings of IEEE International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–5, 2012.

[18] The MITRE Corporation. https://attack.mitre.org/, 2018.

[19] Gideon Creech and Jiankun Hu. Generation of a new ids test dataset: Time to retire the kdd collection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 4487–4492. IEEE, 2013.

[20] Priyanka Dahiya and Devesh Kumar Srivastava. Network intrusion detection in big dataset using spark. *Procedia computer science*, 132:253–262, 2018.

[21] Abbas Abou Daya, Mohammad A. Salahuddin, Noura Limam, and Raouf Boutaba. A graph-based machine learning approach for bot detection. In *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management*, 2019.

[22] Dorothy Denning and Peter G Neumann. *Requirements and model for IDES-a real-time intrusion-detection expert system*. SRI International, 1985.

[23] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[24] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009.

[25] Sergiu Gatlan. Asus live update infected with backdoor in supply chain attack, Mar 2019. Accessed: 2019-04-05.

[26] Ibrahim Ghafir, Mohammad Hammoudeh, Vaclav Prenosil, Liangxiu Han, Robert Hegarty, Khaled Rabie, and Francisco J Aparicio-Navarro. Detection of advanced persistent threat using machine-learning correlation analysis. *Future Generation Computer Systems*, 89:349–359, 2018.

[27] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[28] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Proceedings of IEEE International*

Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pages 1322–1328, 2008.

[29] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. Accessed Mar 2019.

[30] JPCERT/CC. https://www.jpcert.or.jp/english/pub/sr/ir_research.html, 05 2017.

[31] G. Kaiafas, G. Varisteas, S. Lagraa, R. State, C. D. Nguyen, T. Ries, and M. Ourdane. Detecting malicious authentication events trustfully. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium*, pages 1–6, 2018.

[32] Alexander D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.

[33] Alexander D. Kent. Cybersecurity Data Sources for Dynamic Network Research. In *Proceedings of Dynamic Networks in Cybersecurity*. Imperial College Press, June 2015.

[34] Gisung Kim, Seungmin Lee, and Sehun Kim. A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*, 41(4):1690–1700, 2014.

[35] Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

[36] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[37] Yihua Liao and V Rao Vemuri. Use of k-nearest neighbor classifier for intrusion detection. *Computers & security*, 21(5):439–448, 2002.

[38] Qingyun Liu, Jack W Stokes, Rob Mead, Tim Burrell, Ian Hellen, John Lambert, Andrey Marochko, and Weidong Cui. Latte: Large-scale lateral movement detection. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pages 1–6. IEEE, 2018.

[39] Mirco Marchetti, Fabio Pierazzi, Michele Colajanni, and Alessandro Guido. Analysis of high volumes of network traffic for advanced persistent threat detection. *Computer Networks*, 109:127–141, 2016.

[40] Wes McKinney. Data structures for statistical computing in python. In *Proceedings of the Python in Science Conference*, 2010.

[41] microsoft. https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-0708, 2019.

[42] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[43] Travis E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition, 2015.

[44] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[45] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

[46] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12, 2011.

[47] Leonid Portnoy. *Intrusion detection with unlabeled data using clustering*. PhD thesis, Columbia University, 2000.

[48] E. Rescorla. https://tools.ietf.org/html/rfc8446, 08 2018.

[49] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[50] Sara Sinclair, Sean W. Smith, Stephanie Trudeau, M. Eric Johnson, and Anthony Portera. Information risk in financial institutions: Field study and research roadmap. In Daniel J. Veit, Dennis Kundisch, Tim Weitzel, Christof Weinhardt, Fethi A. Rabhi, and Federico Rajola, editors, *Proceedings of Enterprise Applications and Services in the Finance Industry*, 2008.

[51] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 2019.

[52] TokenPost. Crypto exchange dragonex lost $7m in hack, announces compensation plan, Apr 2019. Accessed: 2019-04-05.

[53] Laura Toloi and Thomas Lengauer. Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics*, 27(14):1986–1994, 2011.

[54] Aaron Randall Tuor, Ryan Baerwolf, Nicolas Knowles, Brian Hutchinson, Nicole Nichols, and Robert Jasper. Recurrent neural network language models for open vocabulary event-level cyber anomaly detection. In *Proceedings of Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[55] Ikram Ullah. Detecting lateral movement attacks through smb using bro. Master's thesis, University of Twente, 2016.

[56] Piet Van Mieghem. The n-intertwined sis epidemic network model. *Computing*, 93:147–169, 2011.

[57] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 408–421, 1972.

[58] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*, pages 21–24, 2016.

[59] Nong Ye, Syed Masum Emran, Qiang Chen, and Sean Vilbert. Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions on computers*, 51(7):810–820, 2002.

APPENDICES

Appendix A

Appendix

A Feature Index Table

ID	Feature	ID	Feature	ID	Feature
1	$ID_{(usr,src)}(dst)$	11	$IDAFSTD_{src}(dst)$	21	$ODS_{(usr,dst)}(src)$
2	$IDAF_{(usr,src)}(dst)$	12	$IDS_{src}(dst)$	22	$OD_{dst}(src)$
3	$IDAFSTD_{(usr,src)}(dst)$	13	$WIDS_{src}(dst)$	23	$ODAF_{dst}(src)$
4	$IDS_{(usr,src)}(dst)$	14	$ID_{usr}(dst)$	24	$ODAFSTD_{dst}(src)$
5	$WIDS_{(usr,src)}(dst)$	15	$IDAF_{usr}(dst)$	25	$ODS_{dst}(src)$
6	$MSF(dst)$	16	$IDAFSTD_{usr}(dst)$	26	$OD_{usr}(src)$
7	$SUR(dst)$	17	$IDS_{usr}(dst)$	27	$ODAF_{usr}(src)$
8	$AS(host)$	18	$OD_{(usr,dst)}(src)$	28	$ODAFSTD_{usr}(src)$
9	$ID_{src}(dst)$	19	$ODAF_{(usr,dst)}(src)$	29	$ODS_{usr}(src)$
10	$IDAF_{src}(dst)$	20	$ODAFSTD_{(usr,dst)}(src)$		

B Build Authentication Dictionary

Algorithm 2 Build the $InHostUserMap$ dictionary

```
input  : Authentication log $AuthLog$
output: $InHostUserMap$ dictionary representing the number of incoming authentication
        events recorded at each host per username per day
begin $BuildInHostUserMap(AuthLog)$
    $InHostUserMap \leftarrow dict\{\}$ for $event \in AuthLog$ do
        if $event_{dst} \notin InHostUserMap$ then
            $InHostUserMap[event_{dst}] \leftarrow dict\{\}$
        end
        if $event_{usr} \notin InHostUserMap[]$ then
            $InHostUserMap[event_{dst}][event_{usr}] \leftarrow dict\{\}$
        end
        $day \leftarrow event_{time}$ / 86400
        if $day \notin InHostUserMap[event_{dst}][event_{usr}]$ then
            $InHostUserMap[event_{dst}][event_{usr}][day] \leftarrow 0$
        end
        $InHostUserMap[event_{dst}][event_{usr}][day] \leftarrow$
             $InHostUserMap[event_{dst}][event_{usr}][day] + 1$
    end
    return $InHostUserMap$
end
```

Algorithm 3 Build the $InHostSrcMap$ dictionary

```
input  : Authentication log $AuthLog$
output: $InHostSrcMap$ dictionary representing the number of incoming authentication
        events recorded at each host per source host per day
begin $BuildInHostSrcMap(AuthLog)$
    $InHostSrcMap \leftarrow dict\{\}$  for each event $\in AuthLog$ do
        if $event_{dst} \notin InHostSrcMap$ then
            $InHostSrcMap[event_{dst}] \leftarrow dict\{\}$
        end
        if $event_{src} \notin InHostSrcMap[event_{dst}]$ then
            $InHostUserMap[event_{dst}][event_{src}] \leftarrow dict\{\}$
        end
        $day \leftarrow event_{time}$ / 86400
        if $day \notin InHostSrcMap[event_{dst}][event_{src}]$ then
            $InHostSrcMap[event_{dst}][event_{src}][day] \leftarrow 0$
        end
        $InHostSrcMap[event_{dst}][event_{src}][day] \leftarrow$
          $InHostSrcMap[event_{dst}][event_{src}][day] + 1$
    end
    return $InHostSrcMap$
end
```

Algorithm 4 Build the $InHostUsrSrcMap$ dictionary

```
input  : Authentication log $AuthLog$
output: $InHostUsrSrcMap$ dictionary representing the number of incoming authentica-
        tion events recorded at each host per (username, source host) per day
begin $BuildInHostUsrSrcMap(AuthLog)$
    $InHostUsrSrcMap \leftarrow dict\{\}$  for each event $\in AuthLog$ do
        if $event_{dst} \notin InHostUsrSrcMap$ then
            $InHostUsrSrcMap[event_{dst}] \leftarrow dict\{\}$
        end
        if $event_{(usr,src)} \notin InHostUsrSrcMap[event_{dst}]$ then
            $InHostUsrSrcMap[event_{dst}][event_{(usr,src)}] \leftarrow dict\{\}$
        end
        $day \leftarrow event_{time}$ / 86400
        if $day \notin InHostUsrSrcMap[event_{dst}][event_{(usr,src)}]$ then
            $InHostUsrSrcMap[event_{dst}][event_{(usr,src)}][day] \leftarrow 0$
        end
        $InHostUsrSrcMap[event_{dst}][event_{(usr,src)}][day] \leftarrow$
          $InHostUsrSrcMap[event_{dst}][event_{(usr,src)}][day] + 1$
    end
    return $InHostUsrSrcMap$
end
```

Algorithm 5 Build the $OutHostUsrMap$ dictionary

```
input  : Authentication log $AuthLog$
output: $OutHostUsrMap$ dictionary representing the number of outgoing authentication
        events recorded at each host per username per day
begin $BuildOutHostUsrMap(AuthLog)$
    $OutHostUserMap \leftarrow dict\{\}$ for $each\ event \in AuthLog$ do
        if $event_{src} \notin OutHostUserMap$ then
            $OutHostUserMap[event_{src}] \leftarrow dict\{\}$
        end
        if $event_{usr} \notin OutHostUserMap[event_{src}]$ then
            $OutHostUserMap[event_{src}][event_{usr}] \leftarrow dict\{\}$
        end
        $day \leftarrow event_{time}$ / 86400
        if $day \notin OutHostUserMap[event_{src}][event_{usr}]$ then
            $InHostUserMap[event_{src}][event_{usr}][day] \leftarrow 0$
        end
        $InHostUserMap[event_{src}][event_{usr}][day] \leftarrow$
          $InHostUserMap[event_{src}][event_{usr}][day] + 1$
    end
    return $OutHostUsrMap$
end
```

Algorithm 6 Build the *OutHostDstMap* dictionary

```
input  : Authentication log $AuthLog$
output: $OutHostDstMap$ dictionary representing the number of outgoing authentication
        events recorded at each host per destination host per day
begin $BuildOutHostDstMap(OutHostDstMap)$
    $OutHostDstMap \leftarrow dict\{\}$  for each event $\in AuthLog$ do
        if $event_{src} \notin OutHostDstMap$ then
            $OutHostDstMap[event_{src}] \leftarrow dict\{\}$
        end
        if $event_{dst} \notin OutHostDstMap[event_{src}]$ then
            $OutHostDstMap[event_{src}][event_{dst}] \leftarrow dict\{\}$
        end
        $day \leftarrow event_{time}$ / 86400
        if $day \notin OutHostDstMap[event_{src}][event_{dst}]$ then
            $OutHostDstMap[event_{src}][event_{dst}][day] \leftarrow 0$
        end
        $OutHostDstMap[event_{src}][event_{dst}][day] \leftarrow$
          $OutHostDstMap[event_{src}][event_{dst}][day] + 1$
    end
    return $OutHostDstMap$
end
```

Algorithm 7 Build the $OutHostUsrDstMap$ dictionary

```
input  : Authentication log $AuthLog$
output: $OutHostUsrDstMap$ dictionary representing the number of outgoing authentication events recorded at each host per (username, destination host) per day
begin $BuildOutHostUsrDstMap(OutHostUsrDstMap)$
    $OutHostUsrDstMap \leftarrow dict\{\}$ for $each\ event \in AuthLog$ do
        if $event_{src} \notin OutHostUsrDstMap$ then
            $OutHostUsrDstMap[event_{src}] \leftarrow dict\{\}$
        end
        if $event_{(usr,dst)} \notin OutHostUsrDstMap[event_{src}]$ then
            $OutHostUsrDstMap[event_{src}][event_{(usr,dst)}] \leftarrow dict\{\}$
        end
        $day \leftarrow event_{time}$ / 86400
        if $day \notin OutHostUsrDstMap[event_{src}][event_{(usr,dst)}]$ then
            $OutHostUsrDstMap[event_{src}][event_{(usr,dst)}][day] \leftarrow 0$
        end
        $OutHostUsrDstMap[event_{src}][event_{(usr,dst)}][day] \leftarrow$
          $OutHostUsrDstMap[event_{src}][event_{(usr,dst)}][day] + 1$
    end
    return $OutHostUsrDstMap$
end
```

C Generating Features from Dictionaries

The following algorithms describe all the features in section A

Algorithm 8 Calculate $ID_{usr}(dst_j)$

```
input  : $InHostUserMap$, $dst_j$
output: $ID_{usr}(dst_j)$
if $dst_j\ in\ InHostUserMap$ then
    return $number\ of\ usr\ in\ InHostUserMap[dst_j]$
else
    return 0
end
```

Algorithm 9 Calculate $ID_{src}(dst_j)$

```
input  : $InHostSrcMap$, $dst_j$
output: $ID_{src}(dst_j)$
if $dst_j$ in $InHostSrcMap$ then
    return number of src in $InHostSrcMap[dst_j]$
else
    return 0
end
```

Algorithm 10 Calculate $ID_{(usr,src)}(dst_j)$

```
input  : $InHostUsrSrcMap$, $dst_j$
output: $ID_{(usr,src)}(dst_j)$
if $dst_j$ in $InHostUsrSrcMap$ then
    return number of $(usr, src)$ in $InHostUsrSrcMap[dst_j]$
else
    return 0
end
```

Algorithm 11 Calculate $OD_{usr}(src_j)$

```
input  : $OutHostUserMap$, $src_j$
output: $OD_{usr}(src_j)$
if $hostnameinOutHostUserMap$ then
    return number of usr in $OutHostUserMap[src_j]$
else
    return 0
end
```

Algorithm 12 Calculate $OD_{dst}(src_j)$

```
input  : $OutHostDstMap$, $src_j$
output: $OD_{dst}(src_j)$
if $src_j$ in $OutHostDstMap$ then
    return number of dst in $OutHostDstMap[src_j]$
else
    return 0
end
```

Algorithm 13 Calculate $OD_{(usr,dst)}(src_j)$

```
input  : $OutHostUsrDstMap$, $src_j$
output: $OD_{(usr,dst)}(src_j)$
if $src_j$ in $OutHostUsrDstMap$ then
    return number of $(usr, dst)$ in $OutHostUsrDstMap[src_j]$
else
    return 0
end
```

Algorithm 14 Calculate $IDAF_{usr}(dst_j)$

```
input  : $InHostUsrMap$, $dst_j$
output: $IDAF_{usr}(dst_j)$
$UserFrequency \leftarrow list[]$ if $dst_j$ in $InHostUsrMap$ then
    for each username in $InHostUsrMap[dst_j]$ do
        $UsrFreq \leftarrow$
            Average logon times per day for username $UserFrequency.append(UsrFreq)$
    end
    return the average of $UserFrequency$
else
    return 0
end
```

Algorithm 15 Calculate $IDAF_{src}(dst_j)$

```
input  : $InHostSrcMap$, $dst_j$
output: $IDAF_{src}(dst_j)$
$SrcFrequency \leftarrow list[]$ if $dst_j$ in $InHostSrcMap$ then
    for each src in $InHostSrcMap[dst_j]$ do
        $SrcFreq \leftarrow$
            Average logon times per day for src $SrcFrequency.append(SrcFreq)$
    end
    return the average of $SrcFrequency$
else
    return 0
end
```

Algorithm 16 Calculate $IDAF_{(usr,src)}(dst_j)$

```
input  : $InHostUsrSrcMap$, $dst_j$
output: $IDAF_{(usr,src)}(dst_j)$
$UsrSrcFrequency \leftarrow list[]$  if $dst_j$ in $InHostUsrSrcMap$ then
    for each usr-src in $InHostUsrSrcMap[dst_j]$ do
        $UsrSrcFreq \leftarrow$
          Average logon times per day for usr-src $UsrSrcFrequency.append(UsrSrcFreq)$
    end
    return the average of $UsrSrcFrequency$
else
    return 0
end
```

Algorithm 17 Calculate $ODAF_{usr}(src_j)$

```
input  : $OutHostUsrMap$, $src_j$
output: $ODAF_{usr}(src_j)$
$UserFrequency \leftarrow list[]$  if $src_j$ in $OutHostUsrMap$ then
    for each username in $OutHostUsrMap[src_j]$ do
        $UsrFreq \leftarrow$
          Average logon times per day for username $UserFrequency.append(UsrFreq)$
    end
    return the average of $UserFrequency$
else
    return 0
end
```

Algorithm 18 Calculate $ODAF_{dst}(src_j)$

```
input  : $OutHostDstMap$, $src_j$
output: $ODAF_{dst}(src_j)$
$DstFrequency \leftarrow list[]$ if $src_j$ in $OutHostDstMap$ then
    for each dst in $OutHostDstMap[src_j]$ do
        $DstFreq \leftarrow$
          Average logon times per day for dst $DstFrequency.append(DstFreq)$
    end
    return  the average of $DstFrequency$
else
    return 0
end
```

Algorithm 19 Calculate $ODAF_{(usr,dst)}(src_j)$

```
input  : $OutHostUsrDstMap$, $src_j$
output: $ODAF_{(usr,dst)}(src_j)$
$DstFrequency \leftarrow list[]$ if $src_j$ in $OutHostUsrDstMap$ then
    for each usr-dst in $OutHostUsrDstMap[src_j]$ do
        $UsrDstFreq \leftarrow$
          Average logon times per day for usr-dst $UsrDstFrequency.append(UsrDstFreq)$
    end
    return the average of $UsrDstFrequency$
else
    return 0
end
```

Algorithm 20 Calculate $IDAFSTD_{usr}(dst_j)$

```
input  : $InHostUsrMap$, $dst_j$
output: $IDAFSTD_{usr}(dst_j)$
$UserFrequency \leftarrow list[]$  if $dst_j$ in $InHostUsrMap$ then
    for each username in $InHostUsrMap[dst_j]$ do
        $UsrFreq \leftarrow$
          Average logon times per day for username $UserFrequency.append(UsrFreq)$
    end
    return the standard deviation of $UserFrequency$
else
    return 0
end
```

Algorithm 21 Calculate $IDAFSTD_{src}(dst_j)$

```
input  : $InHostSrcMap$, $dst_j$
output: $IDAFSTD_{src}(dst_j)$
$SrcFrequency \leftarrow list[]$  if $dst_j$ in $InHostSrcMap$ then
    for each src in $InHostSrcMap[dst_j]$ do
        $SrcFreq \leftarrow$
          Average logon times per day for src $SrcFrequency.append(SrcFreq)$
    end
    return the standard deviation of $SrcFrequency$
else
    return 0
end
```

Algorithm 22 Calculate $IDAFSTD_{(usr,src)}(dst_j)$

```
input  : $InHostUsrSrcMap$, $dst_j$
output: $IDAFSTD_{(usr,src)}(dst_j)$
$UsrSrcFrequency \leftarrow list[]$ if $dst_j$ in $InHostUsrSrcMap$ then
    for each usr-src in $InHostUsrSrcMap[dst_j]$ do
        $UsrSrcFreq \leftarrow$
            Average logon times per day for usr-src $UsrSrcFrequency.append(UsrSrcFreq)$
    end
    return the standard deviation of $UsrSrcFrequency$
else
    return 0
end
```

Algorithm 23 Calculate $ODAFSTD_{usr}(src_j)$

```
input  : $OutHostUsrMap$, $src_j$
output: $ODAFSTD_{usr}(src_j)$
$UserFrequency \leftarrow list[]$ if $src_j$ in $OutHostUsrMap$ then
    for each username in $OutHostUsrMap[src_j]$ do
        $UsrFreq \leftarrow$
            Average logon times per day for username $UserFrequency.append(UsrFreq)$
    end
    return the standard deviation of $UserFrequency$
else
    return 0
end
```

Algorithm 24 Calculate $ODAFSTD_{dst}(src_j)$

```
input  : OutHostDstMap, $src_j$
output: $ODAFSTD_{dst}(src_j)$
$DstFrequency \leftarrow list[]$ if $src_j$ in $OutHostDstMap$ then
    for each dst in $OutHostDstMap[src_j]$ do
        $DstFreq \leftarrow$
          Average logon times per day for dst $DstFrequency.append(DstFreq)$
    end
    return the standard deviation of $DstFrequency$
else
    return 0
end
```

Algorithm 25 Calculate $ODAFSTD_{(usr,dst)}(src_j)$

```
input  : OutHostUsrDstMap, $src_j$
output: $ODAFSTD_{(usr,dst)}(src_j)$
$DstFrequency \leftarrow list[]$ if $src_j$ in $OutHostUsrDstMap$ then
    for each usr-dst in $OutHostUsrDstMap[src_j]$ do
        $UsrDstFreq \leftarrow$
          Average logon times per day for usr-dst $UsrDstFrequency.append(UsrDstFreq)$
    end
    return the standard deviation of $UsrDstFrequency$
else
    return 0
end
```

Algorithm 26 Sparseness function (*SF*)

```
input  : Source host $Src$, username $Usr$, destination host $Dst$, thresholds $\theta$, $\beta$
output: $Sparseness$, a sparseness score of event defined by $Src$, $Usr$, and $Dst$
Initialize $Events$ to all events in authentication log  $Sparseness \leftarrow 0$
 /* filter(*) is a no-op, $countByDays()$ counts the numbers of days where the events occur
 */
$TotalDays \leftarrow Events.filter(Src, Usr, Dst).countByDays()$
if $TotalDays \leq \theta$ then
    $Sparseness \leftarrow max(TotalDays * \beta - Events.count(), 0)$
end
return $Sparseness$
```

Algorithm 27 Calculate $IDS_{usr}(dst_j)$

```
input  : $InHostUserMap$, $dst_j$
output: $IDS_{usr}(dst_j)$
$UserSparses \leftarrow list[]$  if $dst_j$ in $InHostUserMap$ then
    for each usr in $InHostUserMap[dst_j]$ do
        $UserSparse \leftarrow SF(*, usr, dst_j, \theta, \beta)$
        $UserSparses.append(UserSparse)$
    end
    return the sum of $UserSparses$
else
    return 0
end
```

Algorithm 28 Calculate $IDS_{src}(dst_j)$

```
input  : $InHostSrcMap$, $dst_j$
output: $IDS_{src}(dst_j)$
$SrcSparses \leftarrow list[]$ if $dst_j$ in $InHostSrcMap$ then
    for each src in $InHostSrcMap[dst_j]$ do
        $SrcSparse \leftarrow SF(src, *, dst_j, \theta, \beta)$
        $SrcSparses.append(SrcSparse)$
    end
    return the sum of $SrcSparses$
else
    return 0
end
```

Algorithm 29 Calculate $IDS_{(usr,src)}(dst_j)$

```
input  : $InHostUsrSrcMap$, $dst_j$
output: $IDS_{(usr,src)}(dst_j)$
$UsrSrcSparses \leftarrow list[]$ if $dst_j$ in $InHostUsrSrcMap$ then
    for each usr-src in $InHostUsrSrcMap[dst_j]$ do
        $UsrSrcSparse \leftarrow SF(src, *, dst_j, \theta, \beta)$
        $UsrSrcSparses.append(UsrSrcSparse)$
    end
    return the sum of $UsrSrcSparses$
else
    return 0
end
```

Algorithm 30 Calculate $ODS_{usr}(src_j)$

```
input  : OutHostUserMap, $src_j$
output: $ODS_{usr}(src_j)$
$UserSparses \leftarrow list[]$ if $src_j$ in $OutHostUserMap$ then
    for each usr in OutHostUserMap[$src_j$] do
        $UserSparse \leftarrow SF(src_j, usr, *, \theta, \beta)$
        $UserSparses.append(UserSparse)$
    end
    return the sum of $UserSparses$
else
    return 0
end
```

Algorithm 31 Calculate $ODS_{dst}(src_j)$

```
input  : OutHostDstMap, $src_j$
output: $ODS_{dst}(src_j)$
$DstSparses \leftarrow list[]$ if $src_j$ in $OutHostDstMap$ then
    for each dst in OutHostDstMap[$dst_j$] do
        $DstSparse \leftarrow SF(src_j, *, dst, \theta, \beta)$
        $DstSparses.append(DstSparse)$
    end
    return the sum of DstSparses
end
return 0
```

Algorithm 32 Calculate $ODS_{(usr,dst)}(src_j)$

```
input  : OutHostUsrDstMap, $src_j$
output: $ODS_{(usr,dst)}(src_j)$
$UsrDstSparses \leftarrow list[]$ if $src_j$ in $OutHostUsrDstMap$ then
    for each (usr,dst) in $OutHostUsrDstMap$[$src_j$] do
        $UsrDstSparse \leftarrow SF(src_j, usr, dst, \theta, \beta)$
        $UsrDstSparses.append(UsrDstSparse)$
    end
    return the sum of $UsrDstSparses$
end
return 0
```

Algorithm 33 Calculate $WIDS_{src}(dst_j)$

```
input  : InHostSrcMap, $dst_j$
output: $WIDS_{src}(dst_j)$
$WeightedSrcSparses \leftarrow list[]$ if $dst_j$ in $InHostSrcMap$ then
    for each src in $InHostSrcMap[dst_j]$ do
        $SrcSparse \leftarrow SF(src, *, dst_j, \theta, \beta)$
        $Weight \leftarrow ODS_{dst}(src)$
        $WeightedSrcSparses.append(SrcSparse * Weight)$
    end
    return the sum of $WeightedSrcSparses$
else
    return 0
end
```

Algorithm 34 Calculate $WIDS_{(usr,src)}(dst_j)$

```
input  : InHostUsrSrcMap, $dst_j$
output: $WIDS_{(usr,src)}(dst_j)$
$WeightedUsrSrcSparses \leftarrow list[]$ if $dst_j$ in $InHostUsrSrcMap$ then
    for each (usr,src) in $InHostUsrSrcMap[dst_j]$ do
        $Weight \leftarrow ODS_{(usr,dst)}(src)$
        $UsrSrcSparse \leftarrow SF(src, *, dst_j, \theta, \beta)$
        $WeightedUsrSrcSparses.append(UsrSrcSparse * Weight)$
    end
    return the sum of $WeightedUsrSrcSparses$
end
return 0
```

Algorithm 35 Calculate $MSF(dst_j)$

```
input  : $InHostSrcMap$, $dst_j$
output: $MSF(dst_j)$
$MSF \leftarrow 0$  if $dst_j$ in $InHostUsrSrcMap$ then
    for each src in $InHostSrcMap[dst_j]$ do
        if $MSF < ODS_{dst}(src)$ then
            $MSF = ODS_{dst}(src)$
        end
    end
    return $MSF$
end
return 0
```

Algorithm 36 Calculate $SUR(dst_j)$

```
input  : $InHostUsrMap$, $dst_j$
output: $SUR(dst_j)$
$NumUsr \leftarrow 0$  $NumSparseUsr \leftarrow 0$  if $dst_j$ in $InHostUsrMap$ then
    for each usr in $InHostUsrMap[dst_j]$ do
        $NumUsr = NumUsr + 1$ if $SF(*, usr, dst_j, \theta, \beta) > 0$ then
            $NumSparseUsr = NumSparseUsr + 1$
        end
    end
    if $NumUsr > 0$ then
        return $NumSparseUsr/NumUsr$
    else
        return 0
    end
else
    return 0
end
```

Algorithm 37 Calculate $AS(dst_j)$

```
input  : $MSFdst_j$, $SUR(dst_j)$
output: $AS(dst_j)$
return $MSFdst_j * SUR(dst_j)$
```