

Signature-based Intrusion Detection Using Imbalanced Datasets

Yue Leng

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Aberdeen.



Department of Computing Science

2023

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2023

Abstract

To ensure the security of systems, intrusion detection is an essential technique in the domain of cyber security. However, developing effective intrusion detection systems is a challenging task that requires precise analysis, simulation of real-life anomalies, and large, updated datasets. This study aims to address this challenge by exploring feasible methods for building signature-based intrusion detection systems and optimizing the effectiveness of limited data. Utilizing the Threat Hunter Playbook and compatible datasets, the study constructed a signature-based intrusion detection system. The study proposes a practical approach that incorporates decision trees and Deep Neural Networks for intrusion detection classification, as well as synthetic data generation techniques using both Generative Adversarial Networks and oversampling methods to handle imbalanced datasets. This study presents an effective methodology for constructing signature-based intrusion detection systems when faced with inadequate and imbalanced datasets. It offers valuable insights into developing efficient and effective intrusion detection systems with limited data.

Acknowledgements

I would like to express my heartfelt gratitude and appreciation to all those who made this paper possible.

I would like to thank the University of Aberdeen which provided me with the High-performance computing platform and other resources to conduct this project. I would also like to extend my sincere thanks to my honours project tutor, Dr Matthew Collinson, for his guidance, encouragement, and insights throughout this entire project, from the initial stages to the completion of the paper. His expertise in the field was invaluable, and his feedback and suggestions helped me to refine my work further. I would also like to thank Dr Huan Yang for his time and advice during this project. His knowledge and experience have enlightened me and inspired some aspects of the project's design. I am also grateful to my personal tutor, Dr Bruno Yun, for the support and encouragement he has shown me throughout my time at the University of Aberdeen. Lastly, I would like to thank my friends and family for their constant love, encouragement, and support.

I couldn't have completed this paper without the invaluable support and guidance provided by all of these individuals and organizations. Thank you all for helping me to bring this project to fruition.

Contents

1	Introduction	10
1.1	Overview	10
1.2	Motivation	11
1.3	Objectives	11
2	Background and Related Work	13
2.1	Intrusion Detection	13
2.2	Host-based and Network-based Intrusion Detection	13
2.3	Signature-based and Anomaly-based Intrusion Detection	14
2.4	Decision Tree	14
2.5	Deep Neural Network	15
2.6	Oversampling	16
2.6.1	Random Oversampling	17
2.6.2	Synthetic Minority Oversampling Technique	17
2.6.3	Adaptive Synthetic Oversampling	17
2.7	Generative Adversarial Network	17
2.8	High-performance Computing	18
3	Threat Hunter Playbook	20
3.1	Security Datasets	20
3.2	Analytics in Hunts	22
3.3	Labeling	22
3.4	Features used in Datasets	23
4	Methodology	25
4.1	Run Jupyter Notebook on HPC	26
4.1.1	First time set up	26
4.1.2	Running notebook	26
4.2	Preprocessing	27
4.2.1	Label Encoder	27
4.2.2	Nomalization	28
4.2.3	Splitting Training and Testing Sets	28
4.3	Resampling Data	29
4.3.1	Random Oversampling	29

4.3.2	Synthetic Minority Oversampling Technique	29
4.3.3	Adaptive Synthetic Oversampling	30
4.4	Synthetic Data Generation	31
4.4.1	Framework of Data Generation	31
4.4.2	Mode-specific Normalization	32
4.4.3	Conditional Generator	33
4.4.4	Network Structure	34
4.5	Decision Tree	34
4.5.1	Criteria	34
4.5.2	Implementation	35
4.6	Deep Neural Network	35
4.6.1	Network Structure	35
4.6.2	Balanced accuracy	36
4.6.3	Implementation	37
5	Results and Evaluation	38
5.1	Evaluation Method	38
5.2	Scenario 1	39
5.3	Scenario 2	41
5.4	Scenario 3	42
5.5	Scenario 4	44
5.6	Scenario 5	45
6	Conclusion, Discussion and Future Work	47
6.1	Conclusion	47
6.2	Discussion	48
6.2.1	Insufficient data in Threat Hunter Playbook and Security Datasets for building strong SIDS	48
6.2.2	Decision Tree performs better than DNN in SIDS binary classification . .	48
6.2.3	CTGAN as an Effective Solution for Handling Imbalanced Datasets in SIDS	49
6.3	Future Work	49
A	User Manual	51
A.1	Running on HPC	51
A.1.1	Register an HPC account	51
A.1.2	Install and Run F5 Network VPN	51
A.1.3	Install and Run PuTTY	52
A.1.4	Request for Allocated Resources	53
A.1.5	Prepare Anaconda Environment	53
A.1.6	Build an SSH Tunnel	54
A.2	Running Jupyter Notebook	55

B	Maintenance Manual	56
B.1	Installation	56
B.2	Software Dependencies	56
B.3	Running the System	57
B.4	Requirements	57
B.5	Key File Paths	57
B.6	Directions for Future Improvements	58
B.6.1	Datasets	58
B.6.2	Preprocessing Data	58
B.6.3	Synthetic Data	58
B.6.4	DNN Classification	59
B.6.5	Others	59
B.7	Bug Reports	59
C	Features Used in Datasets	61
D	Datasets	64
E	Data Missing in Datasets	67

List of Tables

3.1	Numbers of positive and negative samples in datasets	23
5.1	Performance metrics of decision trees on the original datasets: Negative Precision, Negative Recall, Positive Precision, and Positive Recall	39
5.2	Decision trees on oversampled datasets. Data in each cell are "negative precision/negative recall/positive precision/positive recall", "x" indicates that methods cannot be used.	41
5.3	SMOTE results of datasets using DNN classifier with different numbers of hidden layers. Data in each cell are "negative precision/negative recall/positive precision/positive recall".	43
5.4	Performance metrics of decision trees on datasets with synthetic data: Negative Precision, Negative Recall, Positive Precision, and Positive Recall	44
5.5	Synthetic datasets using DNN classifier with different numbers of hidden layers. Data in each cell are "negative precision/negative recall/positive precision/positive recall".	45
B.1	Software dependencies and their usage in the program	56
C.1	Features used in datasets.	61

List of Figures

2.1	A simple neural network structure of DNN (Figure adopted from [21].)	16
2.2	A basic structure of GAN	18
3.1	Percentages of NaN of features in datasets for "LSASS Memory Read Access"	21
4.1	5 scenarios for different datasets handling	25
4.2	An example of label encoding	27
4.3	The schematic of SMOTE (Figure adopted from [14])	30
4.4	A CTGAN example of dataset for "Active Directory Root Domain Modification for Replication Services"	32
4.5	A example of mode-specific normalization (Figure adopted from [38])	33
5.1	Decision tree on the dataset for "Registry Modification for Extended NetNTLM Downgrade"	40
5.2	Confusion Matrixes of ROS, SMOTE and ADASYN on datasets for "Remote WMI ActiveScriptEventConsumers" from left to right	42
A.1	Access maxlogin1 via f5 networks VPN	52
A.2	PuTTY configuration	52
A.3	PuTTY successfully logged in	53
A.4	Being allocated resources successfully	53
A.5	Run Jupyter Notebook Successfully	54
A.6	Example of Jupyter Notebook	55

Chapter 1

Introduction

The central goal of this chapter is to furnish the reader with an introduction to the research, comprising its structure, motivations, and aims. More specifically, the primary function of this chapter is to acquaint the reader with the key themes and goals of the research paper. This will be achieved by contextualizing the research problem and highlighting its significance. Additionally, this chapter will outline the specific research objectives and questions that the paper aims to explore, thus empowering the reader to understand the anticipated contributions of this research.

1.1 Overview

In the field of cyber security, intrusion detection is a crucial technique that ensures the security of systems. With the increasing volume of information and the rise in cyber-attacks worldwide, developing efficient intrusion detection systems has become an imperative task. Despite its significance, intrusion detection remains a complex issue in cyber security due to various factors, including the requirements of large datasets, precise analysis, and simulation of real-life anomalies. In this research, our objective is to propose a feasible approach to building an intrusion detection system.

Specifically, we examine the effectiveness of the Threat Hunter Playbook and its compatible datasets for signature-based intrusion detection on hosts, as well as the validity of the Playbook itself. Additionally, we explore various approaches to handling imbalanced datasets, such as resampling techniques and synthetic data generation using Generative Adversarial Networks (GANs). For intrusion detection classification, we utilize decision trees and Deep Neural Networks (DNNs). By comparing the performance of different techniques for handling imbalanced data and different classification methods, we found that using GANs to synthesize data and utilizing decision trees to classify suspicious actions and normal events produce superior results on our datasets. Therefore, our research provides a methodology for building signature-based intrusion detection with inadequate and imbalanced datasets, which may be common situations in cyber security and other fields. Additionally, this research offers insights into how to take full advantage of limited data.

To provide a clear and concise outline of our research, this paper's structure is as follows: Chapter 2 will provide backgrounds on the methodologies used in this research, including previous related work that inspired our design. Chapter 3 will give a detailed description of the Threat Hunter Playbook, including the analytical rules, datasets used for signature-based intrusion detection, the data labeling method and the features we used in the datasets. In Chapter 4, we will discuss all the significant techniques used in the research, including high-performance computing,

resampling techniques, synthetic data generation method, decision trees, and DNNs. In addition, we will provide the main workflows and ideas of how the research was conducted. Chapter 5 will present all the results obtained, which will be analyzed and discussed in Chapter 6. Moreover, potential future improvements will also be explored in Chapter 6.

1.2 Motivation

Kumar et al. highlighted the necessity of implementing an intrusion detection system in all systems to ensure security [20]. He notes that there is always a tradeoff between security and convenience of use, strict operation controls may ensure system secrecy and integrity, but it comes at the cost of convenience and usefulness. Additionally, access control mechanisms provided by computing systems can only reject illegal access, making it possible for users with permission to engage in illegal information flow. This highlights the importance of addressing insider threats or compromise of the authentication module.

Furthermore, Sawyer et al. found that analysts' fatigue is an unavoidable problem, leading to analysts' decreased accuracy in detecting security events over time [32]. Intrusion detection systems become an essential tool in addressing this problem, filtering out normal events and highlighting suspicious actions. This significantly decreases the workload for analysts, allowing them to focus on more critical events with higher accuracy.

Moreover, integrating machine learning techniques with intrusion detection systems has increased in popularity due to its ability to analyse large datasets quickly, effectively and accurately. Machine learning systems process vast amounts of data with greater speed and accuracy than traditional rule-based systems, ultimately improving the performance of intrusion detection systems.

However, imbalanced data is a common issue that can significantly affect the performance of machine learning algorithms in intrusion detection systems. This is because real-world scenarios often involve more normal actions than suspicious actions. In this context, the scarcity of data concerning malicious actions compared to legitimately normal actions in the dataset may cause the machine learning system to generate a biased model. This results in inaccurate intrusion detection.

The Threat Hunter Playbook is also an example of imbalanced datasets suffering from the nature of cyber security attacks, which causes traditional machine learning algorithms to classify normal events far more accurately than anomalous ones. Therefore, effectively handling imbalanced data is an unavoidable and critical step in constructing an intrusion detection system.

1.3 Objectives

The primary objectives of this research project are threefold:

- To validate the usability of the Threat Hunter Playbook and its compatible Security Datasets.
- To determine the feasibility of incorporating machine learning techniques into signature-based intrusion detection systems.
- To tackle the significant challenge of handling extremely imbalanced datasets.

The primary focus of this research project is to validate the usability of the Threat Hunter Playbook and its compatible datasets, the Security Dataset. The playbook provides various analytics according to the datasets, and primarily implements signature-based intrusion detection

through pattern matching. Due to the ever-evolving methods of intrusion and the unprecedented speed of system updates, intrusion detection requires constant updating of the pattern matching rules and datasets. The Threat Hunter Playbook and its datasets are one of the few publicly available host-based signature-based detection tools, and they have been available since 2018, with updates until September 2022. However, given the newness of these tools, very few studies have utilized them. Additionally, the datasets have no labels, and we label data by simply applying analytics in the playbook on them, without using any other anomaly detection tools. This labeling method poses certain ambiguities that require investigation with respect to its usability and effectiveness. From this study, we aim to determine the feasibility of using the Threat Hunter Playbook and its datasets by analyzing their performance in intrusion detection systems and validating the labeling method.

The second objective of this research project is to investigate the potential of machine learning techniques in incorporating signature-based intrusion detection systems in our datasets. With the exponential increase in data, traditional signature-based intrusion detection methods have limitations that require the integration of machine learning techniques. Our research aims to determine the feasibility of using machine learning algorithms, such as decision trees and deep neural networks, in intrusion detection systems by comparing their performance with traditional signature-based systems. By analyzing the results of these experiments, we can identify the potential of machine learning approaches in improving the efficacy of intrusion detection systems.

Lastly, the most significant and challenging objective of this research project is to develop appropriate methods for handling imbalanced datasets, which is common in the area of cybersecurity. Imbalanced datasets represent a significant challenge in intrusion detection system design since rare events might hold vital information that can improve the detection of new attacks. Conversely, imbalanced data may also lead to the generation of biased models as the system focuses on detecting more common events than the few that are actually of interest. Dealing with imbalanced datasets with few samples can be difficult as collecting more samples may not always be possible or feasible. As a result, it becomes essential to explore various approaches to handling imbalanced datasets effectively. Consequently, our research aims to investigate various techniques such as oversampling techniques, as well as more advanced synthetic data generation techniques using Generative Adversarial Networks (GANs).

Overall, by fulfilling the 3 objectives we mentioned above, we can take full advantage of the data we have, even though it is imbalanced, to improve the performance of signature-based intrusion detection systems in real-world situations. At the same time, we can also discover critical information regarding the effectiveness of each approach, which can be used in improving the design of future intrusion detection systems with limited data.

Chapter 2

Background and Related Work

This chapter provides an overview of the fundamental concepts related to the research and discusses the methods that were used in further investigation. Additionally, this chapter references many previous studies in the field, which contribute to the basic conceptual structure and framework of this research and are consulted when designing the experiments.

2.1 Intrusion Detection

The idea of Intrusion Detection was first introduced by James Anderson in 1980, defining an intrusion attempt, or a threat as a potential unauthorized attempt to access or manipulate information or compromise system reliability [2]. Since then, various intrusion detection techniques have been developed and studied. In 1987, the first intrusion detection system model was proposed by Dorothy Denning of Georgetown University and Peter Neumann of SRI/CSL [8].

The purpose of Intrusion Detection Systems (IDS) is to detect potential system breaches caused by malicious actors through automated means. This issue has been extensively studied in both academic research and industry, owing to the increasing significance of security threats [29].

2.2 Host-based and Network-based Intrusion Detection

There are many intrusion detection approaches for detecting and preventing malicious activities in computer systems. In terms of their input source, intrusion detection systems can be classified into three types: Host-based Intrusion Detection System (HIDS), Network-based Intrusion Detection System (NIDS), and Hybrid Intrusion Detection System. While NIDS monitors network traffic to collect input data, HIDS focuses on monitoring the activity of individual computers and servers and collects input data from the host it monitors. A Hybrid Intrusion Detection System, on the other hand, combines both approaches by collecting input data from both network traffic and the hosts it monitors [26].

Host-based intrusion detection systems (HIDS) typically operate by monitoring the internals of the systems such as system logs and file changes, as well as network connections and processes running on the host. HIDS can detect attacks that may be missed by network-based intrusion detection systems (NIDS), such as attacks that originate from a compromised host within the network. On the other hand, NIDS are designed to monitor network traffic for signs of malicious activity. NIDS can detect attacks that are aimed at multiple hosts on the network, such as network scans, port scans, and denial-of-service attacks [25, Ch. 10] .

Both HIDS and NIDS have their advantages and disadvantages, and the choice of which

approach to use depends on the specific security requirements and characteristics of the system being protected.

2.3 Signature-based and Anomaly-based Intrusion Detection

In IDSs research, there are two primary detection methods: anomaly-based and signature-based (a.k.a. misuse detection). Anomaly detection involves detecting intrusions by monitoring for unusual behavior and resource usage on a computer system, while misuse intrusion detection involves defining intrusions beforehand and monitoring for their specific occurrences [20].

IDSs utilizing anomaly-based approaches rely on models of baseline or normal system behavior to identify and report unusual events [29]. The process of identifying anomalous states in a system in the anomaly detection approach is based on detecting significant differences in the system's state transitions from its normal states [33]. Anomaly detection methods use statistical analysis, artificial neural network technology, data mining technology, and artificial immune technology, among others, to detect abnormal behavior. They can identify brand-new attacks that have never been seen before, but they have higher false positive rates compared to signature-based methods [26]. Anomaly-based techniques have been employed in system call sequences as well as in other forms of intrusion detection [29].

Signature-based intrusion detection, which relies on pre-defined patterns or signatures of known attacks, is the primary alternative to anomaly-based IDS and one of the most commonly used methods in both HIDS and NIDS. Signature-based approaches function in a similar way to antivirus scanners as they identify and report events that match the signature of a previously known attack. One example is the MITRE ATT&CK Framework [34], which is a collection of signatures presented as rules for detecting intrusions that may be used to signal events for further scrutiny. Signature-based IDS typically have a high level of efficiency and low false-positive rates. However, they are limited to detecting known attacks because they rely on detecting a sequence of activities that matches a known attack signature [29]. Methods of expert system, TCP/IP protocol analysis, and pattern matching are commonly used in misuse intrusion detection [26].

If an attack has occurred in the past and its signature is known to the intrusion detection system, its recurrence will trigger a misuse detection. On the other hand, an attack that has never been seen before will trigger an anomaly detection [1]. The choice of intrusion detection methods should be based on system requirements, and many security systems use a combination of multiple intrusion detection mechanisms.

2.4 Decision Tree

Decision tree analysis is a technique that uses a divide-and-conquer approach for classification and regression. It is a type of supervised machine learning algorithm and a popular method for decision-making, predictive modeling, and data mining. A decision tree consists of a hierarchical structure in which internal nodes represent features or attributes, branches represent decisions, and leaf nodes represent the outcome or target variable. The construction of the tree involves recursively partitioning the feature space of the training set and dividing the dataset into smaller subsets based on the values of attributes. The objective is to minimize impurity, increase the homogeneity of the subsets, and obtain a set of decision rules that can effectively partition the space and create an informative and reliable hierarchical classification model [23].

Various scoring criteria have been created to assess decision tree partitions, with two main ones being Gini Index (Gini) [4] and Information Gain (InfoGain) [27]. There are also different algorithms to build decision trees, such as the ID3, C4.5, CART, and CHAID algorithms [27], each with their own advantages and limitations. Decision trees are easy to interpret and visualize, however, they also can suffer from overfitting, instability, and bias.

The power of decision trees to identify significant attributes for efficient partitioning and decision-making based on large databases makes them a natural fit for signature-based intrusion detection. Decision trees can enhance the efficiency of signature-based detection by avoiding the sequential comparison of each input event to all rules. A system applying decision trees for a network-based IDS was implemented by Kruegel and Tothin 2003 [19], their experimental evaluation shows that the speed of the detection process has been significantly improved.

Research by Rai, Devi and Guleria developed a decision tree algorithm based on the C4.5 approach [28], focusing on feature selection and split value determination. The algorithm selects the most relevant features using information gain and ensures that the classifier is unbiased towards frequent values when selecting the split value. They conducted experiments on the NSL-KDD (Network Security Laboratory Knowledge Discovery and Data Mining) dataset with varying numbers of features and analyzed the time taken by the classifier to construct the model and the achieved accuracy. The results show that the proposed Decision Tree Split (DTS) algorithm is suitable for signature-based intrusion detection.

2.5 Deep Neural Network

Given that the brain is the most effective machine we have for problem-solving and learning, it is natural to seek inspiration from it when developing machine learning techniques. The concept behind neural networks is derived from the idea that a neuron in our brains performs a computation that involves the weighted sum of input values. These weighted sums represent the scaling of values by synapses and their combination within the neuron. Then a functional operation is carried out within the neuron on the combined inputs. This functional operation is typically a non-linear function that enables a neuron to produce an output only when the inputs exceed a certain threshold [35].

A regular 3-layer neural network is illustrated diagrammatically in 2.1. The input layer neurons receive some values and transmit them to the middle layer of the network, also known as the "hidden layer". The input is transformed through a sequence of hidden layers, each of which comprises a group of neurons. The weighted sums from all the hidden layers are ultimately transmitted to the output layer, which provides the final outputs of the network to the user. Each neuron in a hidden layer is connected to all neurons in the previous layer, and neurons in the same layer function independently without sharing connections [21]. The final layer is known as the "output layer" and is fully connected. While deep neural networks (DNNs) are neural networks that consist of more than three layers, specifically more than one hidden layer, and thus have a more complex architecture and are capable of learning more intricate features and patterns in data [35].

DNNs have revolutionized many fields such as image and speech recognition [18; 7], achieving state-of-the-art performance in a wide range of tasks. Also, due to their capacity to learn from

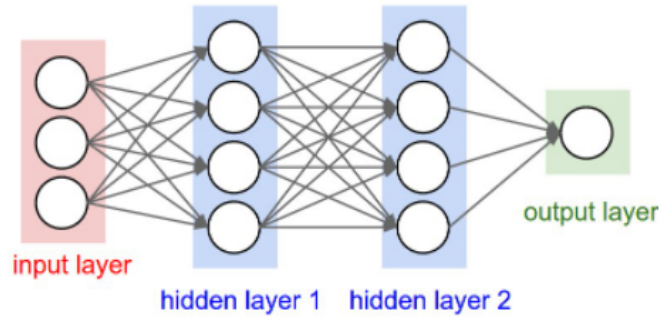


Figure 2.1: A simple neural network structure of DNN (Figure adopted from [21].)

vast amounts of data, enabling accurate predictions on previously unseen data, DNNs are well-suited for use in intrusion detection. McElwee et al. developed an intrusion detection system that utilizes a DNN classifier, which was thoroughly examined and assessed in their study [22]. The research findings indicated that the deep neural network classifier achieved exceptional accuracy by identifying the heuristics utilized by cyber defense security analysts during their analysis. Furthermore, demonstrations and interviews with security analysts showed that their prototype with the DNN classifier was capable of efficiently categorizing security alerts.

Kim et al. conducted a study on an AI-based intrusion detection system that utilized a deep neural network (DNN) for detecting network attacks that are constantly evolving [16]. The DNN algorithm was applied to the preprocessed data to generate a learning model, which was tested on the entire KDD Cup 99 dataset [9], a dataset developed by the Defense Advanced Research Projects Agency (DARPA) by simulating intrusions in a military network environment. And good results of the DNN model were found in terms of accuracy, detection rate, and false alarm rate for intrusion detection.

2.6 Oversampling

In machine learning, imbalanced data refers to a situation where the number of instances in one class greatly outnumber the instances in another class. This is a common problem in many real-world applications such as fraud detection, medical diagnosis, and anomaly detection as a consequence of the intrinsic of data, i.e., the imbalance is a direct result of the nature of the dataspace [13]. Imbalanced datasets can cause machine learning models to perform poorly, as they may be biased towards the majority class, resulting in poor predictions on the minority class [37]. Undersampling (a.k.a. downsampling) and oversampling (a.k.a. upsampling) are two commonly used techniques for imbalanced data. Undersampling is the process of reducing the number of samples in the majority class to balance the class distribution. Oversampling, on the other hand, is the process of increasing the number of samples in the minority class. Given the small sample size problem in our dataset, discarding potentially useful information from the majority class samples to balance with our extremely small minority class could lead to a loss of information and reduced accuracy. Therefore, in our case, only the oversampling technique is appropriate.

Oversampling involves adding more instances to the minority class by either replicating existing minority class samples or generating synthetic new samples using various techniques. Specifically, advanced oversampling techniques such as random oversampling (ROS), synthetic minority

oversampling technique (SMOTE), and adaptive synthetic oversampling (ADASYN) are commonly employed for addressing imbalanced datasets.

2.6.1 Random Oversampling

Random Over-Sampling (ROS) is a data augmentation technique that addresses class imbalance by balancing the class distribution through the addition of a set of randomly selected minority class examples to the original set [13]. To create new synthetic samples that are similar to the existing minority class samples, but with small variations, ROS oversamples the minority class. The synthetic samples are generated by adding a small amount of noise to a randomly selected minority class sample. This process is repeated until the desired level of oversampling is achieved.

2.6.2 Synthetic Minority Oversampling Technique

Synthetic Minority Over-sampling Technique (SMOTE) is another data augmentation technique used to address imbalanced datasets, it is first proposed by Chawla et al [5]. Unlike ROS, SMOTE generates synthetic samples by interpolating between minority class samples. For each minority class sample, SMOTE selects one or more nearest neighbors and generates synthetic samples by linearly interpolating between the original sample and its neighbors. This creates new synthetic samples that are similar to the existing minority class samples, but are not exact copies.

While SMOTE can be effective in addressing class imbalance, it has been observed that in some cases, the synthetic samples generated by SMOTE may be too close to the majority class samples. This can lead to overgeneralization of the decision boundary and reduce the classification performance on the minority class [36]. This issue can be particularly problematic when the minority class is highly overlapped with the majority class.

2.6.3 Adaptive Synthetic Oversampling

Adaptive Synthetic Sampling (ADASYN) is a variation of SMOTE that adaptively generates synthetic samples according to the degree of class imbalance. The SMOTE algorithm can potentially generate synthetic samples that are too similar to the original minority examples, leading to overgeneralization. This is because SMOTE generates the same number of synthetic samples for each original minority example, without considering the neighboring examples. As a result, the classes may overlap, causing decreased performance [36].

To address this issue, Adaptive Synthetic Sampling (ADASYN) algorithms are proposed [12], it is an enhanced version of SMOTE with a minor modification. In ADASYN, more synthetic samples are generated for minority class samples that are harder to learn by the classifier, and fewer synthetic samples are generated for minority class samples that are easier to learn. This approach allows ADASYN to better balance the class distribution than SMOTE in cases where the degree of class imbalance is high.

2.7 Generative Adversarial Network

Generative Adversarial Networks (GANs) are an artificial intelligence algorithm created to address the generative modeling problem, the algorithm generates additional examples from the estimated probability distribution [11]. GANs have the potential to learn versatile representations that can be applied in a range of tasks such as image synthesis, semantic image editing, style transfer, image superresolution, and classification [6].

The basic structure of GANs is shown in the 2.2, it is based on the idea of training two neural networks at the same time: a generator network that produces new data samples, and a discriminator network that assesses the authenticity of the generated samples. The generator network is taught to produce samples that can deceive the discriminator network, while the discriminator network is taught to correctly classify the samples as real or fake. This adversarial training allows the generator network to improve its ability to create more realistic samples, while the discriminator network becomes better at distinguishing between real and fake samples.

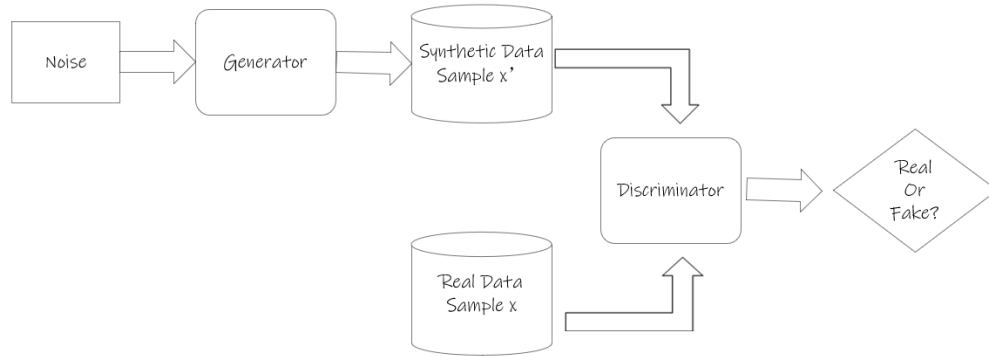


Figure 2.2: A basic structure of GAN

Conditional Tabular Generative Adversarial Network (CTGAN) is a type of GAN specifically designed for generating synthetic tabular data that can preserve the statistical properties of the original dataset. It was proposed in 2019 by Xu et al [38]. CTGAN differs from traditional GANs that aim to generate continuous data, such as images, as it is specifically designed to handle discrete or categorical data typically present in tabular datasets. The approach it utilizes is called "conditional generation," where the generator is conditioned on a specific set of input features to generate data that meets certain conditions.

In terms of preserving the statistical characteristics of the initial dataset when generating synthetic tabular data, CTGAN has been shown to outperform other state-of-the-art methods including Bayesian networks, and it has been demonstrated that the conditional generator can be used for data augmentation [38]. By training a GAN on the original data, CTGAN can comprehend the fundamental data distribution and create new synthetic data that closely approximates the original data, which makes it particularly useful when the available data is limited or sensitive.

2.8 High-performance Computing

High Performance Computing (HPC) is a multidisciplinary field that combines digital electronics, computer architecture, system software, programming languages, algorithms, and computational techniques to utilize supercomputers, computer clusters, or parallel processing techniques to solve complex computational problems or process large amounts of data in a timely manner [3]. It involves the use of specialized hardware and software for intensive calculations, simulations, and modeling tasks that are beyond the capabilities of conventional computing systems.

Due to the computational demands of our research exceeding the limitations of our own devices, we have decided to utilize the HPC resources provided by the University of Aberdeen for a portion of our experiments.

Secure Shell (SSH) is a widely-used network protocol that allows users to securely access and manage remote computer systems. When using SSH to connect to an HPC system, we are essentially establishing a secure connection between our local computer and the remote server, allowing us to run applications and execute commands on the HPC system. By leveraging the computing power of the HPC system, we can then perform computationally-intensive tasks and process large amounts of data in a timely manner.

Chapter 3

Threat Hunter Playbook

The Threat Hunter Playbook is an open source (Jupyter Notebooks) project that aims to make the development of detection logic more efficient by sharing detection strategies, adversary techniques, and resources with pre-recorded security datasets [30]. It follows the structure of MITRE ATT&CK, categorizing post-compromise adversary behavior into tactical groups.

Our research is built upon the foundation of the Threat Hunter Playbook and its corresponding datasets. In this study, we aim to investigate the hunting techniques and hypotheses presented in the playbook and assess their effectiveness in detecting potential security threats. Also, by leveraging the resources and techniques provided in the playbook, we validated the analytics and discuss the usability of the pre-recorded datasets and assessed their suitability for our research purposes.

3.1 Security Datasets

The pre-recorded datasets used by Threat Hunter Playbook are Security Datasets [31], which consist of datasets for different systems including Server, Linux and Windows. Security Datasets is an open-source project that provides a collection of security-related datasets for research and development purposes. The project aims to help security researchers, data scientists, and analysts to evaluate and test their algorithms, models, and techniques on real-world datasets. The datasets provided by Security Datasets cover a wide range of security-related topics, including malware analysis, intrusion detection, network traffic analysis, and more. They are available in various formats, including CSV, PCAP, and JSON, and can be used for different purposes, such as training machine learning models, testing and benchmarking intrusion detection systems, and analyzing network traffic patterns.

The Security Datasets contribute malicious and benign datasets by simulating various specific host-based and network-based attacks from adversaries. Specifically, Security Datasets provides a variety of Windows datasets that can be used for security research and testing. These datasets include various artifacts and logs generated by Windows operating systems, such as event logs, process information, registry keys, and network traffic.

The Windows datasets are designed to simulate various attack scenarios and malicious activities, such as lateral movement, privilege escalation, and data exfiltration. The detailed descriptions and simulation methods for datasets used by hunting programs are given in Appendix D. They can be used to evaluate the effectiveness of security tools and techniques, as well as to develop and test new detection methods. Some examples of the available Windows datasets include the Windows

Event Forwarding dataset, which contains forwarded events from multiple Windows endpoints, the Windows Security Dataset, which contains security-related logs and events, and the Windows Sysmon dataset, which contains data collected by the Sysmon tool.

However, out of the numerous datasets available, the analytics in Threat Hunter Playbook utilize only 21 of them. As a result, our research focuses solely on these 21 datasets, disregarding the rest. By narrowing our scope, we can devote our attention to thoroughly investigating the relevant datasets, ensuring the accuracy and validity of our findings. This also allows us to effectively analyze the data and obtain meaningful insights, without the unnecessary distractions of irrelevant datasets.

One major challenge that we encountered with the datasets is the prevalence of missing data. For instance, consider the datasets for "LSASS Memory Read Access." Apart from the data for features such as "Channel," "EventID," and "timestamp," the datasets have only small portions of data for other features. Out of the 6062 samples, there are 5993 samples without "ObjectName," 5740 samples without "SubjectUserName," 5754 samples without "TargetImage," 5945 samples without "ImageLoaded," 5619 samples without "ProcessGuid," and 5754 samples without "SourceProcessGUID." Figure 3.1 captures the percentages of data missing (considered as NaN in the datasets) in this dataset. Similarly, in nearly every other dataset (as shown in Appendix E), we observed a severe issue of missing data. Missing data can have a profound impact on machine learning models, particularly when datasets have a large proportion of missing data.

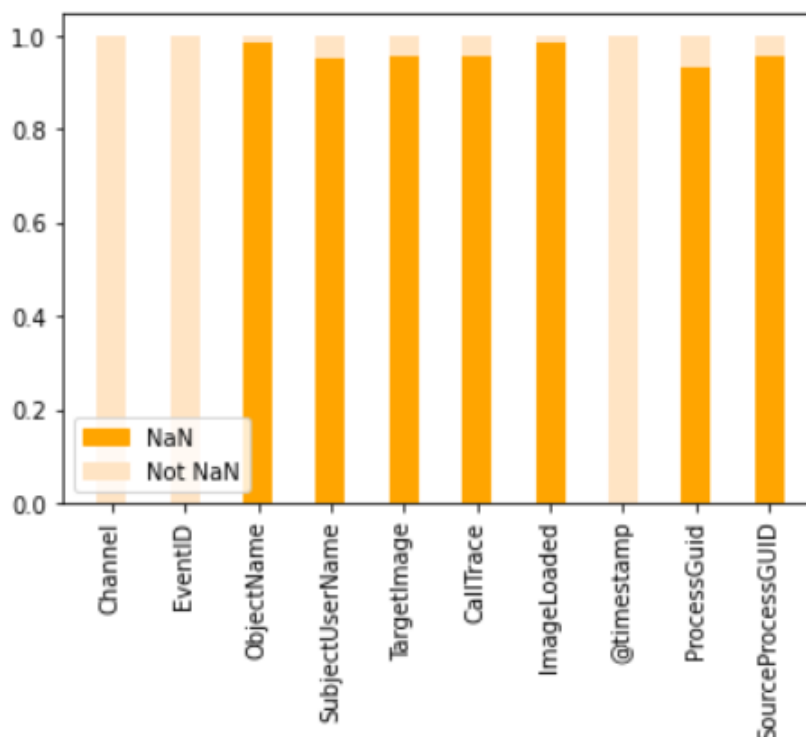


Figure 3.1: Percentages of NaN of features in datasets for "LSASS Memory Read Access"

3.2 Analytics in Hunts

The analytics in Threat Hunter Playbook are a collection of detection methods, also known as "hunts", that are designed to detect various types of adversary behavior. These hunts are created by the community and are based on real-world use cases, threat intelligence, and research. Each hunt typically contains a set of queries or rules that are executed against data sets to identify specific patterns or indicators of compromise. The approaches of pattern matching used by hunts are conceptually the same as the methods used in signature-based intrusion detection.

For instance, the "LSASS Memory Read Access" hunting program in the playbook is based on the hypothesis that intruders may access the Local Security Authority Subsystem Service (LSASS) and extract credentials from memory. To that end, the data used in this hunt is associated with adversaries reading credentials from the memory contents of "lsass.exe". The playbook presents 4 corresponding analytics, including (I) searching for non-system accounts that obtain a handle to and access lsass, (II) identifying processes that open handles and access Lsass with potential Dynamic Link Libraries (DLLs) in memory (e.g. UNKNOWN in CallTrace), (III) detecting processes that load a few known DLLs used by tools such as Mimikatz to interact with credentials, and (IV) combining some of these analytics [30].

As of April 10, 2023, there are a total of 26 guided hunts provided by the Threat Hunter Playbook. However, during our investigation, we encountered issues with 3 of the hunts. Two of the hunts failed to run due to code errors, while one hunt did not produce any results, rendering it impossible to proceed with the labeling process. As a result, we had to abandon this hunt. Additionally, we discovered that there were overlaps between two of the datasets used by the hunts, which means that different hunts detect different types of intrusions based on the same datasets. Consequently, we focused our investigation on the remaining 23 guided hunts, which utilized a total of 21 datasets.

3.3 Labeling

Labeling in machine learning refers to the process of assigning one or more predefined categories or tags to a dataset. In order to train machine learning models to identify and categorize potentially malicious activity in the security datasets, it is necessary to provide labeled data. To achieve this, we defined "negative" as representing normal, benign behavior and "positive" as representing anomalous, potentially suspicious activity that may require manual inspection in Intrusion Detection System (IDS). As the original security datasets did not contain labels, we used the analytics provided in the Threat Hunter Playbook to create our own labels. We designated all samples that matched the patterns identified by the analytics as positive data, while tagging all other samples as negative data.

After conducting our investigation based on the guided hunts in Threat Hunter Playbook, we obtained the results presented in table 3.1. As can be seen, the majority of the datasets used in the playbook are small in size and highly imbalanced. In fact, some datasets have only a single anomaly sample, while having a large number of negative samples. For example, in the dataset for "DLL Process Injection via CreateRemoteThread" and the dataset for "Local Service Installation", there are only 1 anomaly sample, but 12199 and 4347 negative samples, respectively. Even in the dataset for "Registry Modification for Extended NetNTLM Downgrade", which is the best-case

Hunting Program	Negative	Positive
LSASSMemoryReadAccess	6019	7
DLLProcessInjectionCreateRemoteThread	12199	1
ADObjectAccessReplication	8461	4
ADModDirectoryReplication	8994	8
LocalPwshExecution	2057	10
RemotePwshExecution	2735	9
PwshAlternateHosts	2737	7
DomainDPAPIBackupKeyExtraction	7882	5
RegKeyAccessSyskey	12341	8
RemoteWMIExecution	6381	2
WMIEventing	79890	6
WMIModuleLoad	5893	5
LocalServiceInstallation	4347	1
RemoteServiceInstallation	4346	2
RemoteSCMHandle	4559	11
RemoteInteractiveTaskMgrLsassDump	5486	3
RegModExtendedNetNTLMDowngrade	1613	22
MicrophoneDvcAccess	6192	10
RemoteWMIActiveScriptEventConsumers	3709	10
RemoteDCOMIertUtilDLLHijack	37811	3
RemoteWMIWbemcomnDLLHijack	8944	3
RemoteCreateFileSMB	502	4
WuacltCreateRemoteThread	1322	4

Table 3.1: Numbers of positive and negative samples in datasets

scenario, there are only 22 positive samples and 1613 negative samples.

The severe class imbalance in these datasets poses a significant challenge in the learning process, and may negatively impact the performance of our models. Additionally, the scarcity of useful information in some of these datasets may render them unusable. Therefore, we need to apply various techniques to handle the imbalanced data, to ensure that our models can effectively learn and detect intrusions in the face of such class imbalance.

3.4 Features used in Datasets

In the hunting programs, to facilitate the intrusion detection process, not all the data contained in the security datasets is relevant, only specific features that are potentially essential for detection are selected. For example, in the "LSASS Memory Read Access" hunting program, only a few features such as "Channel", "EventID", "ObjectName", and "SubjectUserName" are utilized in Analytic I. Apart from these, "TargetImage" and "CallTrace" are also used in the matching process of Analytic II, "ImageLoaded" and "@timestamp" are used in Analytic III, and "ProcessGuid" and "SourceProcessGUID" are used in the joining process of Analytic IV. Therefore, only the features we mentioned are left in the dataset. By removing irrelevant data, we can extract the important features from the datasets and make it easier for machine learning models to learn. Additionally, this significantly reduces the computational requirements of the system.

The datasets used for the hunting programs contain a range of key features that are crucial in

identifying attacks. Each hunting program utilizes a varying number of features, ranging from 2 to 13. Some features, such as "Channel" and "EventID," are utilized in all the hunting programs, while others, like "PipeName" and "PrivilegeList," are employed in specific circumstances. The complete list of features employed in all the datasets is available in Appendix C. Each feature is essential and conveys unique information regarding the security event logs in the systems, which can be complicated and is subject to the operating system's specifications.

For example, in Windows-based systems, System Monitor (Sysmon), a Windows system service and device driver, monitors and logs system activity to the Windows event log once installed on the system. It provides detailed information about process creations, network connections, and changes made to file creation times. For instance, in the Sysmon channel, Event 1 represents process creation, Event 2 indicates a process changed a file creation time, while Event 3 shows a network connection. There are also other noteworthy features recorded that are used in generating signature-based rules. However, it is essential to note that networks in our classification understand only the labels we provide and cannot discern the intricacies of each feature.

Chapter 4

Methodology

The initial part of this chapter provides information on how to run Jupyter Notebook on the High-Performance Computing Server of the University of Aberdeen. While this is not directly related to the project, it is essential due to the limitations of devices. If you feel that this part is not useful and not necessary in your case, you can skip it. In the following sections, the methods used in this project are explained, including preprocessing, data handling, and machine-learning model building.

Regarding data handling, two different methods were employed - resampling data and generating synthetic data - to determine which approach is more effective in dealing with imbalanced datasets. Furthermore, when building intrusion detection models, two distinct machine learning techniques were employed: decision tree and Deep Neural Network (DNN) models. This was done to provide a comprehensive comparison of the two approaches. It is important to note that you can replicate some of this research by selecting one data handling method and one machine learning method.

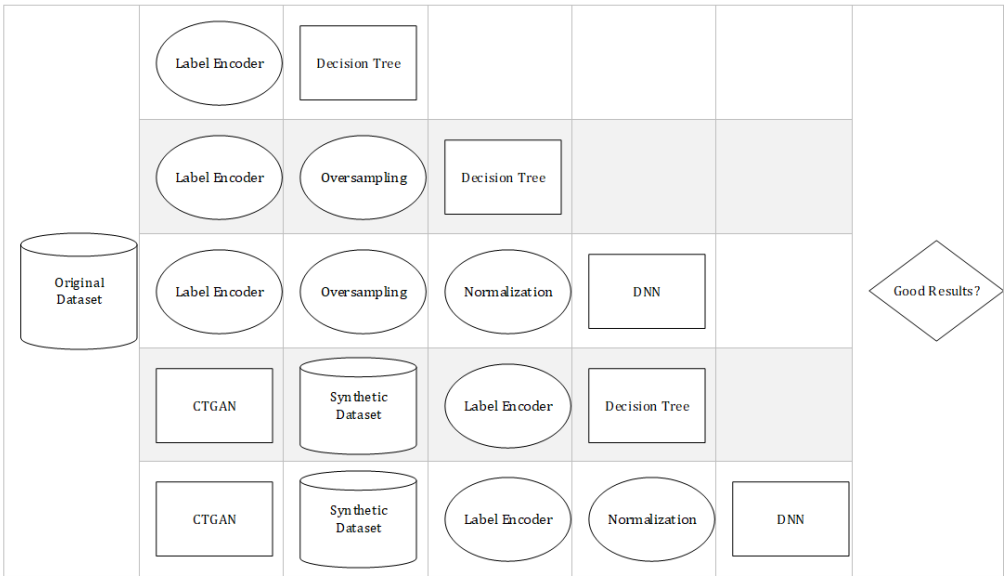


Figure 4.1: 5 scenarios for different datasets handling

Figure 4.1 illustrates the workflow in 5 scenarios that enable the investigation of different datasets. The processes become increasingly complex as we move from the top of the figure to

the bottom. In the best-case scenario, we encode our original data and input it into a decision tree algorithm. If the performance of the original dataset is satisfactory, then no further processing of the dataset is necessary, and we only require the process at the first line of the figure. However, if we cannot obtain good classification results, we must perform further processing on the datasets line by line. In the worst-case scenario, we must go through the processes described in the fifth line of the figure and still cannot achieve good results from the dataset. In this case, we assume that the dataset does not contain adequate information that can be utilized to construct an IDS.

4.1 Run Jupyter Notebook on HPC

To access the High Performance Computing (HPC) system, you need to have access to an HPC server. This usually involves obtaining an account from your institution or from a cloud service provider. Specifically, for HPC of the University of Aberdeen, Maxwell, you can get an account by applying through the school website or contacting the digital research team. Additionally, if you want to access Maxwell on your own devices, you have to connect to the school's VPN first.

4.1.1 First time set up

Once connected to the HPC server, you can start typing commands in your terminal. For the first time, you should create the environment you need. Anaconda is a popular environment used for data analysis and scientific computing, and it is usually installed locally in your account if you are using Maxwell. However, the base version of Anaconda is typically the only version installed in a central location, so you may need to install specific versions of Python depending on your needs. You can install or uninstall these environments without requiring permission or input from a system administrator. In addition, you may also need to configure the notebook environment by installing additional software packages, libraries or dependencies that are required for your specific project. These software packages can be installed using the conda or pip package managers, which are included with the Anaconda environment.

4.1.2 Running notebook

After setting up the Anaconda environment on the server, you can load the module and activate the environment. This can typically be done on the login node, where you can run quick and simple tests. However, for heavy computations, you will need to submit your job to a compute node using the SLURM batch scheduler. To do this, you will need to start an interactive SLURM session to reserve a compute resource.

Once you have reserved a compute resource, you will need to set up SSH port forwarding between the compute resource and your local computer. This can be done by forwarding an unused port on your local machine to the port that Jupyter Notebook is running on the compute resource. This will allow you to access the Jupyter Notebook interface from your local machine via a web browser.

After you start Jupyter Notebook, it will display some basic information, including the URLs for connecting to your notebook. On the client side, in a new terminal window, you can start an SSH tunnel between the server and your local machine. This will prompt you for a password, unless you have set up SSH keys already. Once you have entered the correct password, you can access the Jupyter Notebook interface via your web browser on your local machine by navigating to the corresponding port in your browser.

By following these steps, you can run Jupyter Notebook on an HPC server and access it from your local machine, allowing you to perform heavy computations without being limited by the resources of your local machine.

4.2 Preprocessing

In order to use pre-existing models for machine learning, we will utilize encapsulated libraries such as Scikit-learn for decision tree and Tensorflow for DNN classifier. However, before feeding our data into these models, it is essential to preprocess the input data to ensure that it is in the correct format and to improve the performance of the models. To transform the different types of data into integers, we use label encoders in Scikit-learn. Also, we have experimented with different normalization approaches to see if we can obtain better results with our models. Additionally, due to the limited size of our datasets, we have split the training and testing sets manually. This allows us to evaluate the performance of our models and make necessary adjustments to improve the accuracy of our predictions.

4.2.1 Label Encoder

Label Encoder is a widely-used method for converting categorical variables into numerical format in the field of machine learning. Scikit-learn, a popular machine learning library, provides a built-in LabelEncoder class that can perform label encoding easily. By assigning a unique integer to each category in the variable, the LabelEncoder enables machine learning algorithms to understand the relationships between the categories.

Label encoding is a straightforward and effective technique to convert text data or categorical variables into numerical values that can be fed as input to machine learning models. The LabelEncoder class can be fitted on a categorical variable and then transform the same variable into numerical values. This process is executed for each feature, i.e., each column in the dataset, resulting in a new dataset containing the distribution of the original dataset in numerical format.

For instance, in a dataset for "LSASS Memory Read Access", as shown in the figure 4.2, the feature "Channel" has four classes, and the label encoder would transform them into different numbers, from 0 to the number of classes minus one, according to the occurrence order of the values in the dataset. The transformed dataset, consisting of only numerical values, can be used to build machine learning models such as decision trees, as well as for further investigations.

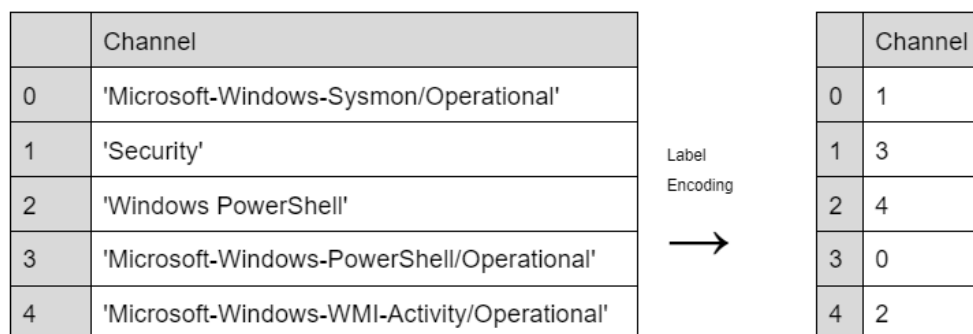


Figure 4.2: An example of label encoding

The LabelEncoder class is a valuable tool in machine learning, especially when dealing with large datasets that contain categorical variables. It simplifies the data preprocessing process, allowing for more efficient model building and evaluation.

4.2.2 Nomalization

Normalization is a data pre-processing technique used to scale the values of a dataset to a specific range. The choice of normalization method depends on the specific algorithm being used and the nature of the data.

Scikit-learn offers several normalization techniques, such as StandarScaler, MinMaxScaler, MaxAbsScaler, RobustScaler, and more. For our dataset, which encodes categorical variables as continuous integers, outliers are not expected, making some normalization methods, like RobustScaler, potentially unnecessary. StandarScaler is used to standardize features by removing the mean and scaling to unit variance, as seen in equation 4.1, where μ represents the mean and σ represents the variance. This method is appropriate when input data features follow a normal distribution. MinMaxScaler scales each feature to a given range, such as between 0 and 1, as shown in equation 4.2, where x_{min} and x_{max} represent the minimum and maximum feature values, respectively. It is recommended when the data is not normally distributed. MaxAbsScaler scales each feature by its maximum absolute value, as seen in equation 4.3, where x_{max} represents the maximum feature value. This normalization method is useful when the data spans a wide range, and it does not rely on normal data distribution.

$$x' = \frac{x - \mu}{\sigma} \quad (4.1)$$

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (4.2)$$

$$x' = \frac{x}{x_{max}} \quad (4.3)$$

For decision trees, when the dataset does not contain features with vastly different scales, normalization is not typically necessary since decision trees are insensitive to the scale of the features. While for Deep Neural Networks (DNN), normalization is an important step in pre-processing the data. Generally, DNNs are sensitive to the scale of input features and can converge faster and achieve better accuracy when the input data is properly normalized. the choice of normalization method depends on the distribution of the input data and the presence of outliers. It is always required to experiment with different normalization methods and choose the one that gives the best performance for the specific model architecture and dataset. By experimenting with the different normalization techniques mentioned above, it is possible to identify the best way to preprocess the data and improve the performance of the DNN.

4.2.3 Splitting Training and Testing Sets

Training and testing set splitting is a common technique used in machine learning to evaluate the performance of a model. In this technique, the available dataset is split into two parts: a training set and a testing set. The model is trained on the training set and then tested on the testing set to measure its performance on unseen data. The general guideline is to use the majority of the data

for training the model and only a small portion for testing. A common split is to use 70-80% of the data for training and the remaining 20-30% for testing.

When dealing with imbalanced datasets, it is common practice to oversample the minority class or use Generative Adversarial Networks (GAN) to generate synthetic data. However, it is crucial to note that oversampling and generating synthetic data should only be done on the training set, and not on the testing set. This is because including rebalanced or synthetic data in the testing set could bias the performance evaluation. To ensure that the training and testing sets are not affected by oversampling or synthetic data generation, it is recommended to perform these techniques after splitting the data into training and testing sets. Additionally, it is important to only modify the data in the training set, and leave the testing set untouched.

However, our datasets may have too few positive samples, only 1 in some cases, which can lead to all positive samples ending up in the training set when randomly splitting the data. This leaves very few or no positive samples to support positive recall and accuracy during evaluation. To address this issue, it may be necessary to break the randomness principle of splitting data sets. In these cases, one approach is to manually place positive samples in the testing set first, and then randomly select negative samples to be included in the testing set. This can ensure that the testing set has a sufficient number of positive samples to accurately evaluate the performance of the model.

4.3 Resampling Data

As previously discussed in Section 2.6, there are three main techniques for oversampling data: random oversampling (ROS), synthetic minority oversampling technique (SMOTE), and adaptive synthesis (ADASYN). To improve the class distribution of our data, we apply all three techniques to oversample our dataset, and then we assess their performance to determine the best technique. We utilized **RandomOverSampler**, **SMOTE** and **ADASYN** from the *imblearn.over_sampling* library to implement these oversampling methods.

By using these techniques, we aim to create a more balanced and representative dataset for training our models. To assess the effectiveness of each technique, we compare their respective performances in terms of classification accuracy, precision, recall, and F1-score. By analyzing these metrics, we can determine which oversampling method is most effective for our dataset and can improve the performance of our models.

4.3.1 Random Oversampling

The principle of ROS, as mentioned in Section 2.6.1, involves replicating samples from the minority class and, in some cases, introducing small variations or noise to create new synthetic samples. Given an original set S consists of a minority class S_{min} and a majority class S_{maj} , a set E is created by randomly selecting $|E|$ samples from the minority class. Accordingly, the class distribution balance of S is adjusted: $|S| = |S_{min}| + |S_{maj}| + |E|$. This approach allows for adjusting the degree of class balance to any desired level and it is easy to understand and implement [13].

4.3.2 Synthetic Minority Oversampling Technique

As described in Section 2.6.2, SMOTE generates artificial data by identifying feature space similarities between existing minority class examples. The algorithm shown as Equation 4.4 is used.

For each example x_i in a subset of the minority class S_{min} , the K-nearest neighbors of x_i are identified as the K elements of S_{min} whose Euclidean distance to x_i is smallest. To create a synthetic sample, one of the K-nearest neighbors \hat{x}_i is randomly selected and a vector is generated by multiplying the difference between x_i and \hat{x}_i with a random number δ , $\delta \in [0, 1]$. This vector is then added to x_i to produce a synthetic instance, which is a point along the line segment connecting x_i and \hat{x}_i [13].

$$x_{new} = x_i + (\hat{x}_i - x_i) \times \delta \quad (4.4)$$

In Figure 4.3, to provide an example illustrating how SMOTE works, consider selecting an instance x_1 from the minority class and identifying its K nearest neighbors (K=5) x_2, x_3, x_4, x_5, x_6 . Synthetic samples a, b, c, d , and e are then generated using the Equation 4.4. It is not difficult to find that the resulting samples within the ellipse boundary can be classified as part of the minority class, while those outsiders like e are more similar to the majority class [14].

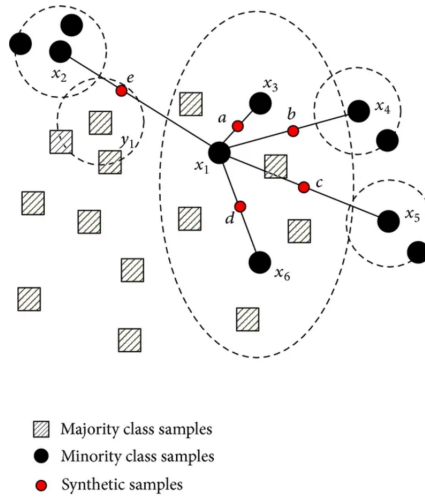


Figure 4.3: The schematic of SMOTE (Figure adopted from [14])

4.3.3 Adaptive Synthetic Oversampling

In Section 2.6.3, it was discussed that ADASYN is an improved version of the SMOTE algorithm, which aims to address its limitations. ADASYN introduces a small modification to the SMOTE algorithm by weighting the creation of synthetic samples for each minority class example. ADASYN works as follows:

First, calculate the total number of synthetic data instances required for the entire minority class:

$$G = (|S_{maj}| - |S_{min}|) \times \beta \quad (4.5)$$

where $|S_{maj}|$ and $|S_{min}|$ represent the number of samples in the majority class and minority class respectively, $\beta \in [0, 1]$, it indicates the desired balance level after the generation process. When it is set to 1, the numbers of samples in the majority class and the minority class are the same after oversampling. Then calculate the ratio of majority examples in K nearest neighbors:

$$\Gamma_i = \Delta_i / K \quad (4.6)$$

where Δ_i is the number of examples belonging to the majority class in K nearest neighbors. After that, normalize Γ_i so that we get $\hat{\Gamma}_i$, and $\sum \hat{\Gamma}_i = 1$:

$$\hat{\Gamma}_i = \Gamma_i / \sum_{i=1}^{|S_{min}|} \Gamma_i \quad (4.7)$$

According to the weights of samples, calculate the number of new synthetic data samples required for each $x_i \in S_{min}$:

$$g = \hat{\Gamma}_i \times G \quad (4.8)$$

Finally, generate g_i synthetic data samples for each $x_i \in S_{min}$ according to Equation 4.4 [13]. Basically, by adjusting the weights of diverse minority instances, ADASYN utilizes a density distribution as a measure to dynamically determine the quantity of synthetic samples required for each minority instance, and thus to counterbalance the skewed distributions. However, it is important to note that ADASYN may also generate synthetic samples that are closer to the majority class, which can negatively affect model performance.

4.4 Synthetic Data Generation

We utilized the **CTGANSynthesizer** provided by Synthetic Data Vault (SDV) to implement CTGAN, a Python library designed for generating tabular synthetic data. The CTGAN model we used is based on the one presented in a paper authored by Xu et al [38]. One of the notable features of this synthesizer is its ability to handle non-numerical columns and those with missing values. Additionally, it automatically converts other data types using Reversible Data Transforms (RDTs). We employed the default value of 300 for the epoch number, which indicates that during the training process, all data is passed through the generator and discriminator 300 times.

4.4.1 Framework of Data Generation

Upon analyzing our dataset, it is evident that the number of negative samples far exceeds the number of positive samples. In some cases, the number of positive samples is too small to provide enough information for oversampling techniques like SMOTE and ADASYN to generate new synthetic samples. Therefore, to overcome this issue, we propose using Conditional Tabular Generative Adversarial Networks (CTGAN) to generate additional synthetic data from the original dataset.

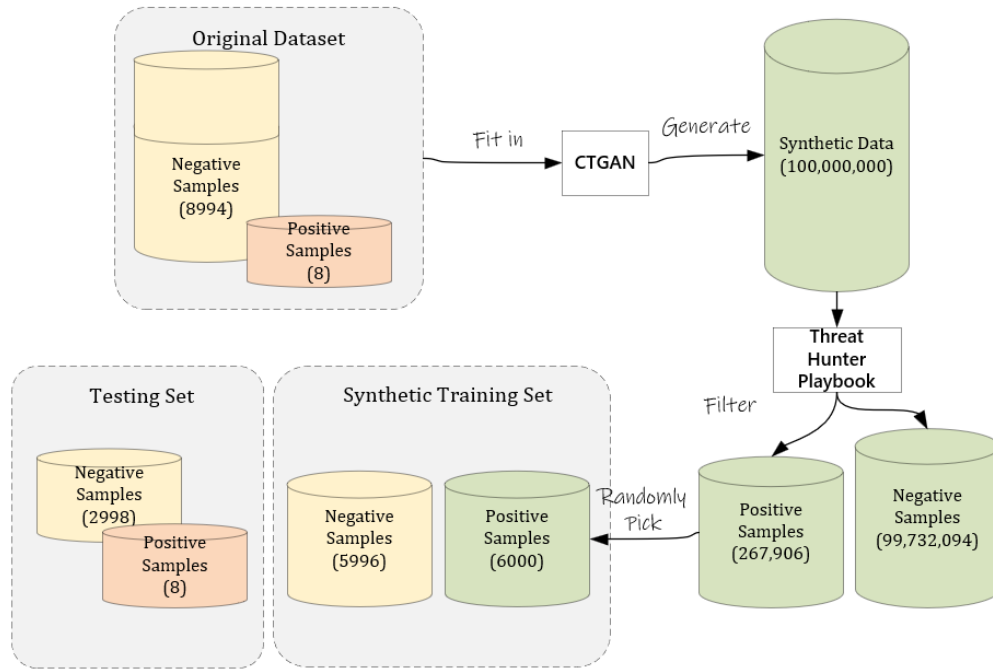


Figure 4.4: A CTGAN example of dataset for "Active Directory Root Domain Modification for Replication Services"

As shown in Figure 4.4, to illustrate our approach, let's consider the "Active Directory Root Domain Modification for Replication Services" dataset from the Threat Hunter Playbook. This dataset has 8994 negative samples and only 8 positive samples. To generate more synthetic data, we feed all the samples from the dataset into the CTGAN without any labels. The CTGAN learns the underlying distribution of the data and generates synthetic samples similar to the original data. After training the CTGAN on the original dataset, we generate 1 million synthetic samples using the model. Since anomaly data is the minority in our dataset, we only need to extract positive samples from the generated synthetic data to rebalance the dataset. To filter the synthetic data, we use analytics from the Threat Hunter Playbook and extract more positive samples.

To prevent bias in our evaluation, the testing set only contains samples from the original dataset. As mentioned in Section 4.2.3, we manually split the original dataset and place 1/3 of the negative samples (i.e. 2998) and all the positive samples (i.e. 8) in the testing set. The remaining 2/3 of the negative samples are used in the training set. To balance the number of negative and positive samples in the training set, we randomly select the same number of positive samples from the synthetic data generated by CTGAN.

In cases where we do not generate enough positive samples within 1 million synthetic data, we can use all the generated positive samples and undersample our negative samples to align with them.

4.4.2 Mode-specific Normalization

Tabular data differs from image data in that it often contains a combination of discrete and continuous columns. Continuous columns may have multiple modes, while discrete columns can be imbalanced, which can make modeling challenging. Traditional statistical and deep neural network models thus may not be suitable for modeling this type of data effectively. In order to better

represent tabular data, Xu et al introduced mode-specific normalization in CTGAN [38], which processes columns independently and represents each value using a one-hot vector and a scalar. The process for normalizing a continuous column using mode-specific normalization is illustrated in Figure 4.5.

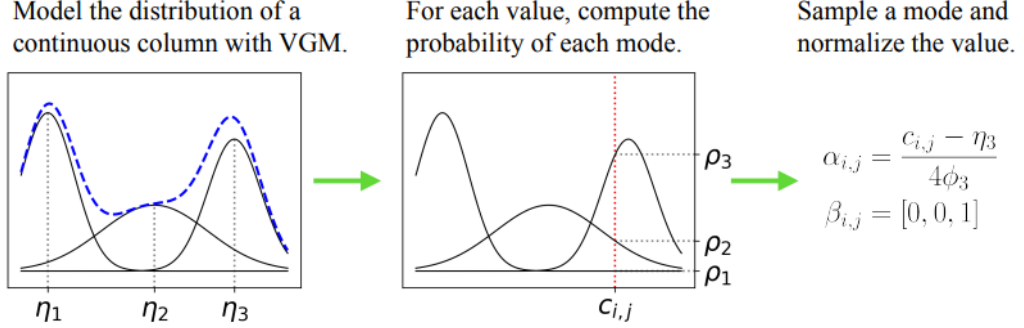


Figure 4.5: A example of mode-specific normalization (Figure adopted from [38])

Variational Gaussian mixture model (VGM) is used to fit a Gaussian mixture for each continuous column C_i . The VGM determines the number of modes ($m_i = 3$) and identifies the modes themselves, such as η_1 , η_2 , and η_3 in Figure 4.5. The resulting Gaussian mixture is represented as $\mathbb{P}_{C_i}(c_{i,j}) = \sum_{k=1}^3 \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$, where μ_k is the weight and ϕ_k represents the standard deviation of a mode. To compute the probability of each value $c_{i,j}$ in C_i from each mode, for example, the probability densities ρ_1 , ρ_2 , and ρ_3 in Figure 4.5, the values are calculated based on the formula $\rho_k = \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$.

Next, one mode is sampled from the probability density, and this mode is used to normalize the value. In Figure 4.5, the third mode η_3 is chosen given the probability densities ρ_1 , ρ_2 , and ρ_3 . To represent $c_{i,j}$, a one-hot vector $\beta_{i,j} = [0, 0, 1]$ is used to indicate the third mode, and a scalar $\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$ is used to represent the value within the mode.

$$r_j = \alpha_{1,j} \oplus \beta_{1,j} \oplus \dots \oplus \alpha_{N_c,j} \oplus \beta_{N_c,j} \oplus d_{1,j} \oplus \dots \oplus d_{N_d,j} \quad (4.9)$$

In this way, each row becomes a concatenation of continuous and discrete columns shown as Equation 4.9, where $d_{i,j}$ represents the one-hot encoding of discrete values. and it allows for the proper representation of tabular data [38].

4.4.3 Conditional Generator

The traditional GAN generator is typically fed with a vector sampled from a standard multivariate normal distribution (MVN). However, the traditional method fails to consider the imbalance in categorical data, as it may not sufficiently represent the rows belonging to the minority category. And to make sure the generator learns the real data distribution, we cannot simply resample data. To address these issues, the generator is then interpreted as the conditional distribution of rows given a particular value at a specific column, and is referred to as the Conditional Generator.

Specifically, we can consider a scenario where we want to generate synthetic data that matches the i^{th} discrete column D_{i*} . Let k^* denote the specific value of D_{i*} that we want to match. In this case, we can interpret the generator as a conditional distribution that generates rows of data given the specific value k^* for the discrete column D_{i*} . We can represent this conditional

distribution as $\mathbb{P}_g(row|D_{i*} = k*)$, where \mathbb{P}_g is the distribution of the synthetic data generated by the CTGAN model.

Xu et al [38] also introduced conditional vector as the way for indicating the condition above ($D_{i*} = k*$), which uses one-hot vectors to represent the discrete columns. And the generator loss is utilized to ensure that the conditional generator generates values that match the specified condition. CTGAN uses the conditional vector to train the generator network to generate synthetic samples that are similar to the original data, while also satisfying the statistical dependencies captured by the conditional vector.

4.4.4 Network Structure

The CTGAN method employs fully-connected networks in both the generator and discriminator, as columns in a row do not have local structure, and thus all possible correlations between the columns must be captured. The generator consists of two fully-connected hidden layers, each followed by batch normalization [15] and Rectified Linear Unit (ReLU) [24] activation function. The synthetic row representation is then generated using a combination of activation functions. The scalar values α_i are generated using the hyperbolic tangent (tanh) activation function, while the mode indicator β_i and discrete values d_i are generated using the Gumbel-Softmax function, which is a continuous approximation to the discrete categorical distribution [38].

On the other hand, the discriminator in CTGAN also employs fully-connected networks with two hidden layers. The leaky ReLU activation function is used in the discriminator to avoid the vanishing gradient problem. Additionally, dropout is applied on each hidden layer of the critic to prevent overfitting [38]. Overall, the CTGAN architecture is designed to effectively capture the complex correlations and dependencies present in the original data while generating synthetic data that is statistically similar to the original data.

4.5 Decision Tree

Once the desired training sets have been produced, either through oversampling or generating synthetic data using CTGAN, a machine learning model is needed to distinguish between normal and anomalous data by training on these sets. As discussed in section 2.4, decision tree algorithms have been widely used in previous studies for intrusion detection, and have been found to achieve high classification accuracy while remaining computationally efficient.

Decision trees offer several advantages over simple pattern matching approaches used in SIDS. For instance, they are capable of extracting important features from the input data and do not require each input to be compared with every rule, resulting in significant time savings and increased efficiency in the detection process. By leveraging the hierarchical structure of decision trees, the algorithm is able to quickly and accurately classify new data points based on the learned patterns from the training set. Therefore, decision tree algorithms should provide a promising approach for detecting anomalous behavior.

4.5.1 Criteria

In decision tree algorithms, different scoring criteria can be used to evaluate and select the best split points for creating tree partitions. Two commonly used criteria are the Gini Index (Gini) [4] and Information Gain (InfoGain) [27]. In our study, we experimented with both scoring methods to determine which one would yield better results for our decision trees.

The Gini Index measures the impurity of a set of examples by calculating the probability of a random sample being incorrectly classified. It ranges from 0, indicating a completely pure set, to 1, indicating a completely impure set (with an even distribution of examples from different classes). The Gini Index for a set is calculated as 1 minus the sum of the squared probabilities p_i of each class in the set. When selecting a split, the goal is to minimize the Gini Index of the resulting subsets.

$$Gini = 1 - \sum_{i=1}^n p_i^2 \quad (4.10)$$

When calculating the Information Gain, we use Entropy as a measure of impurity similar to the Gini Index, but it is calculated as the sum of the negative class probabilities p_i multiplied by their logarithms. Information Gain measures the reduction in entropy achieved by splitting a set of examples on a certain attribute. The Information Gain of a split is calculated as the difference between the entropy of the original set and the weighted average entropy of the resulting subsets as shown in the equation below, where $T|a$ is the conditional entropy of T given the value of attribute a . When selecting a split, the goal is to maximize the Information Gain.

$$Entropy = - \sum_{i=1}^n p_i \times \log_2(p_i) \quad (4.11)$$

$$InfoGain = Entropy(T) - Entropy(T|a) \quad (4.12)$$

By testing both Gini Index and Information Gain in our decision tree models, we aimed to determine which scoring method would produce better results for our specific use case.

4.5.2 Implementation

To apply the decision tree algorithm in our study, we have utilized the **DecisionTreeClassifier** module offered by Scikit-learn. This module facilitates the usage of different criteria for partitioning the decision tree by enabling the modification of the "criterion" parameter of the classifier. By doing so, we can switch between using Gini Index and Information Gain as the scoring criteria for the algorithm. Scikit-learn provides an easy-to-use interface for constructing decision trees, allowing us to experiment with different configurations and hyperparameters. This has enabled us to compare the performance of decision trees using Gini and Information Gain in our study.

4.6 Deep Neural Network

Section 2.5 of this paper mentions that McElwee et al. have developed a Network Intrusion Detection System (NIDS), which utilizes a Deep Neural Network (DNN) classifier [22]. Their research demonstrated that the DNN classifier is capable of detecting network intrusions with exceptional accuracy. The DNN model structure proposed by McElwee et al. is based on the model introduced by Loffe and Szegedy [22].

4.6.1 Network Structure

Our DNN model's architecture is also based on the design proposed by McElwee et al [22]. Their model's architecture includes a variable number of hidden layers between one and five. The choice of the number of hidden layers is a crucial decision in building DNN models as it significantly affects the model's accuracy and computational complexity. Our DNN model's flexible design

allows us to select the optimal number of hidden layers based on the characteristics of the dataset used for training.

The neuron counts for the first hidden layer are determined by multiplying the number of input neurons necessary for the feature vector by three. For example, the analytics in Threat Hunter Playbook for "LSASS Memory Read Access" contain 10 features in total, then the DNN for this dataset should have 30 neurons in the first hidden layer. Subsequent hidden layers have half the number of neurons as the preceding layer. This approach ensures deeper layers contain less information compared to shallower layers, resulting in better performance and faster learning. Using this methodology, a DNN model consisting of 10 input neurons would have hidden nodes of [30, 15, 7, 3] for a four-layered network architecture. All of the hidden layers in the DNN model use the rectified linear unit (ReLU) transfer function [24] as it increases the model's non-linearity and enables it to quickly adjust to changes in the input data.

However, in contrast to the activation function used in the original DNN model proposed by McElwee et al. [22], we have opted to use the sigmoid function instead of the softmax function in our design. The reason for this is that our detection approach is based on a binary classification problem where we only determine whether the input data is an anomaly or not. As a result, our model consists of only two labels, making the use of a softmax function unnecessary. Instead, we utilize a sigmoid function at the output layer of our DNN model, which has only 2 output neurons that correspond to the 2 labels in the training data. This approach allows our model to effectively classify input data into one of the 2 categories, anomaly or not, and ensures accurate and reliable detection of network intrusions.

The DNN model is trained using the adaptive moment estimation (Adam) algorithm [17] due to its efficiency and effectiveness in training deep neural networks. We use the Xavier algorithm [10] for weight initialization which eliminates the problem of vanishing or exploding gradients during training.

By building our DNN model according to the structure proposed by McElwee et al. [22], we can naturally take advantage of the benefits of the design they used to build their NIDS. This approach is cost-effective and saves us the time and resources spent on exploring new DNN architectures. It also ensures that our model is built upon a solid foundation with proven efficacy in network intrusion detection. Furthermore, the architecture proposed by McElwee et al. has been tested and validated for a wide range of datasets, making it a reliable approach to building efficient and effective DNN models in the field of network security.

4.6.2 Balanced accuracy

We use balanced accuracy as the metric when training our DNN models. Balanced accuracy is a metric used to evaluate the performance of a model in classification tasks. It's used when the dataset is imbalanced and the distribution of classes is not equal. In such cases, traditional accuracy metrics can be misleading as they indicate a high level of accuracy even when the model is missing the minor class completely.

Specifically, according to the definition given by TensorFlow, balanced accuracy calculates the average of the true positive rate of each class. The formula for balanced accuracy is shown as Equation 4.13:

$$\text{BalancedAccuracy} = \frac{TPR_1 + TPR_2 + \dots + TPR_N}{N} \quad (4.13)$$

$$TRP = \frac{TP}{TP + FN} \quad (4.14)$$

where TPR (True Positive Rate), as shown in 4.14 is the number of true positives divided by the total number of actual positives for a given class. N is the total number of classes. By calculating each class's true positive rate, balanced accuracy considers the performance of the model across all classes, including the minor one. The value of balanced accuracy ranges between 0 and 1, where a value of 1 indicates that the model has predicted all samples correctly, and 0 indicates that the model has not made any correct predictions.

Overall, balanced accuracy is a more robust metric for evaluating the performance of models in imbalanced datasets, as it ensures that the model's performance is evaluated across all classes, taking into account the imbalanced nature of the datasets.

4.6.3 Implementation

When implementing a Deep Neural Network (DNN) for academic research, selecting the right tools and strategies is essential for building an efficient and high-performing model. In our research, the implementation of a DNN model uses the Keras API in TensorFlow 2.

The number of input features used in the Threat Hunter Playbook analytics determines the input shape definition in the DNN implementation. Additionally, the number of neurons in each hidden layer, the number of output neurons, activation function, weight initializer, and optimizer are specified. Once the parameters are set, the code creates a sequential model using the **Sequential** class in Keras. The input layer is added to the model using the **InputLayer** class, and the subsequent hidden layers are added using **Dense** layers within a loop. The Dense layers create a fully connected layer with the specified number of neurons, activation function, weight initializer, and name. Finally, the model is compiled with the optimizer, loss function, and metrics being specified. The balanced accuracy metric is selected to track during the training phase, ensuring that the model is optimized to produce accurate predictions for both classes, which is significant in the imbalanced datasets.

Overall, the implementation of a DNN involves defining the input shape, selecting the right number of neurons for hidden layers, activation functions, optimizer, and defining the loss function and metrics. Using the right tools and strategies, such as the Keras API, can improve the efficiency and performance of the model, resulting in accurate and reliable predictions.

Chapter 5

Results and Evaluation

The workflow illustrated in Figure 4.1 presents five different scenarios for datasets, where the complexity of the workflow increases as we move from the top to the bottom of the figure. The aim of this workflow is to perform complex processes only on those datasets that contain less useful information and do not demonstrate good classification results with the lower-level workflow.

To present our findings, we structured them according to the 5 different scenarios depicted in Figure 4.1. As we move towards the higher-level scenarios, the number of datasets that require more complex processing reduces. The proposed workflow we adopted follows a strategic approach that ensures efficient utilization of resources and time spent on processing different datasets. By limiting the complexity of the processes based on classification results, we can avoid performing unnecessary operations for specific datasets, leading to a more efficient and cost-effective analysis.

5.1 Evaluation Method

Our Intrusion Detection System (IDS) detects network intrusion based on binary classification of datasets, which leads to evaluating the system's performance solely through classification results on models' testing sets, including their positive recall, positive precision, negative recall, and negative precision. A confusion matrix is also utilized to show the relationship between actual and predicted labels.

For the decision tree algorithm, we implement the classification results directly. However, for the DNN algorithm, we set a prediction cutoff value of 0.5. If the predicted likelihood of one class is higher than this value, we assume that class is the predicted class for the data. Subsequently, we calculate recall and precision scores based on the classification determined.

It's crucial to note that in most IDSs, False Negatives, or the overlooked cases, are more costly than False Alarms or False Positives, meaning that catching more suspicious actions is more important. A high false positive rate is better than a high false negative rate, and a higher recall score is more critical than a higher precision score. The reason lies in the fact that the former situation may bring more work to security analysts, who can verify whether the suspicious action is indeed a threat or not. However, overlooking a legitimate intrusion can lead to catastrophic damage to the system. Thus, the IDS's primary goal needs to be minimizing the number of False Negatives, reducing the chance of an actual intrusion.

5.2 Scenario 1

The first scenario we considered is the best-case scenario, where the dataset doesn't require any specialized handling related to imbalanced data. We can obtain satisfactory classification outcomes by merely using a decision tree algorithm. Before inputting the data into the decision trees, we only mandate a pre-processing process of label encoding.

If the dataset has a good performance in this scenario, it might be due to the dataset already containing enough information for intrusion detection. However, good results might also be the product of inadequate support in the testing set. Unfortunately, there isn't much we can do about the latter since we cannot modify or add samples to the testing set.

This scenario involves a simple pre-processing step of label encoding for the dataset, followed by utilizing the decision tree algorithm. After implementing these steps for all the datasets we have, we obtained the following results:

Hunting Program	Precision(-)	Recall(-)	Precision(+)	Recall(+)
LSASSMemoryReadAccess	1	1	0	0
DLLProcessInjectionCreateRemoteThread	1	1	0	0
ADObjectAccessReplication	1	1	0.75	0.5
ADModDirectoryReplication	1	1	1	0.67
LocalPwshExecution	1	1	0	0
RemotePwshExecution	1	1	0	0.4
PwshAlternateHosts	1	1	0	0
DomainDPAPIBackupKeyExtraction	1	1	0.75	1
RegKeyAccessSyskey	1	1	1	1
WMIEventing	1	1	1	0.5
WMIModuleLoad	1	1	1	0.5
RemoteSCMHandle	1	1	1	0.5
RemoteInteractiveTaskMgrLsassDump	1	1	0	0
RegModExtendedNetNTLMDowngrade	1	1	0.91	1
MicrophoneDvcAccess	1	1	0.33	0.5
RemoteWMIActiveScriptEventConsumers	1	1	0	0
RemoteWMIWbemcomnDLLHijack	1	1	0	0
RemoteCreateFileSMB	1	1	0	0
WuaucItCreateRemoteThread	1	1	0	0

Table 5.1: Performance metrics of decision trees on the original datasets: Negative Precision, Negative Recall, Positive Precision, and Positive Recall

In most of our analyzed datasets, we discovered that using Gini and InfoGain as criteria produced the same results. However, in one dataset, utilizing Entropy as the evaluation method led to slightly better positive recall and negative recall results than Gini. Though this improvement came at a cost of diminishing negative precision and recall. As a result, we appreciate that there is no substantial difference in using different criteria and have resolved to use Gini for our decision tree algorithm.

It's evident that in several of our datasets, the classification of positive samples presents significant challenges. Although all datasets demonstrate perfect recall and precision for negative data, the model struggles to classify positive samples accurately. We're unable to obtain results from datasets with only a few positive samples, like one or two samples, as we don't generate any synthetic data. Putting positive samples in the testing sets renders the model unable to learn, however, adding them to the training sets contributes no support for negative recall and precision during evaluation.

Only datasets for "SysKey Registry Keys Access" and "Registry Modification for Extended NetNTLM Downgrade" generated satisfactory results in the first scenario, with the former achieving perfect 1 for both precision and recall, and the latter scoring 0.91 for positive precision and 1 for positive recall, which we considered satisfactory. For instance, we implemented Gini as the criteria for the decision tree algorithm, generating decision trees on dataset for "Registry Modification for Extended NetNTLM Downgrade", as shown in Figures 5.1.

Decision Tree on the Reg Mod Extended Net NTLM Downgrade

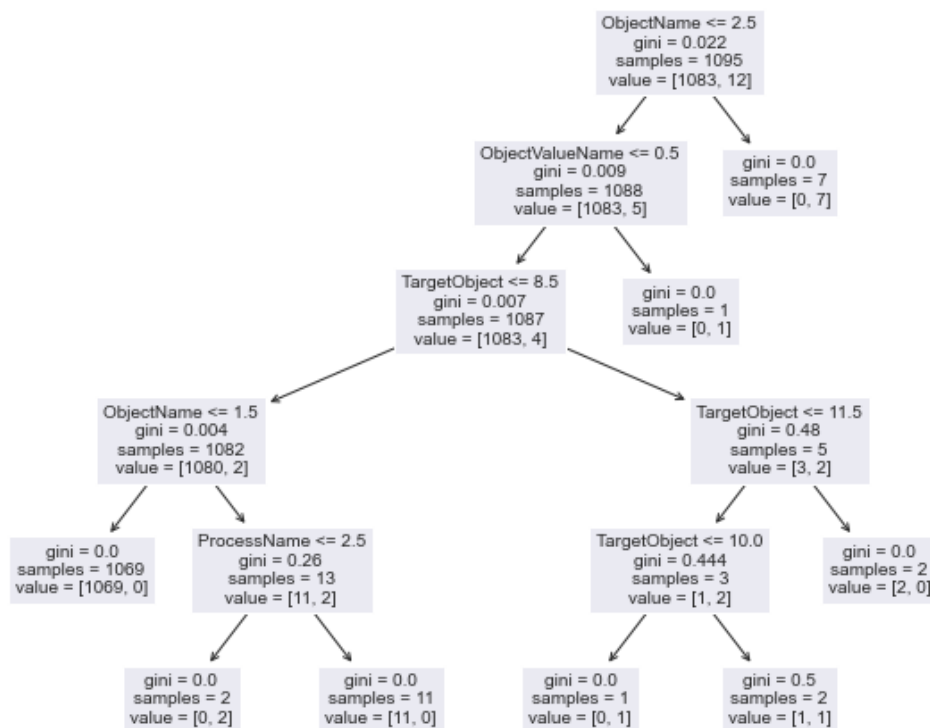


Figure 5.1: Decision tree on the dataset for "Registry Modification for Extended NetNTLM Downgrade"

Among all the analyzed datasets, "Registry Modification for Extended NetNTLM Downgrade" has the most positive samples, containing 22, with 1613 negative samples. This information suggests that the dataset may contain ample information for detecting intrusion. As shown in Figure 5.1, the decision tree generated for this dataset leverages vital features such as "ObjectName", "ObjectValueName", and "TargetObject" for classification, with most of these features resulting in small Gini coefficients. These coefficients indicate that the classified results are relatively pure and that the tree has accurately identified the distinguishing features that can differentiate positive

samples from negative samples, allowing for precise classification of data records.

Similarly, the decision tree generated for the "SysKey Registry Keys Access" dataset mainly uses the "ObjectName" feature for classification, which also plays a crucial role in pattern matching, as outlined in the Threat Hunter Playbook's analysis.

Overall, both the "SysKey Registry Keys Access" and "Registry Modification for Extended NetNTLM Downgrade" datasets exhibit characteristics containing ample information for intrusion detection and they don't require further processing. However, it's worth noting that despite these two datasets containing enough information for intrusion detection, other datasets present significant challenges in accurately classifying positive samples. As discussed previously, we are unable to obtain results from datasets with extremely few positive samples, further emphasizing the need for other solutions to accurately classify data samples.

5.3 Scenario 2

As discussed in section 4.3, in Scenario 2, we resample data after pre-processing them using label encoding and before using the decision tree algorithm. We carry out resampling techniques like ROS, SMOTE, and ADASYN for all the datasets to determine the most suitable oversampling method for each dataset. It's worth noting that resampling must only be conducted on the training data after splitting the training and testing sets to avoid bias. As mentioned earlier, datasets with few positive samples created challenges as putting them in either the training or testing sets led to undesirable classification results.

This scenario involves utilizing oversampling techniques after pre-processing the datasets via label encoding, followed by using the decision tree algorithm, we analyzed our datasets and obtained the following results:

Hunting Program	ROS	SMOTE	ADASYN
LSASSMemoryReadAccess	1/1/0/0	×	×
ADObjectAccessReplication	1/1/0/0	×	×
ADModDirectoryReplication	1/1/1/0.6	×	×
LocalPwshExecution	1/0.96/0/0	1/0.96/0/0	1/0.96/0/0
RemotePwshExecution	1/1/0.67/1	1/1/0.67/1	1/1/0.67/1
PwshAlternateHosts	1/0.94/0/0	1/0.94/0/0	1/0.94/0/0
DomainDPAPIBackupKeyExtraction	1/1/1/1	×	×
WMIEventing	1/1/1/0.5	×	×
WMIModuleLoad	1/1/0.5/1	×	×
RemoteServiceInstallation	1/0.91/0/0	×	×
RemoteSCMHandle	1/1/1/0.5	1/1/0.67/0.5	1/1/0.67/0.5
RemoteInteractiveTaskMgrLsassDump	1/1/0/0	×	×
MicrophoneDvcAccess	1/1/0.25/0.5	1/0.97/0.02/0.5	1/0.97/0.02/0.5
RemoteWMIActiveScriptEventConsumers	1/1/0/0	1/0.99/0.1/1	1/0.99/0.09/1
RemoteDCOMIErtUtilDLLHijack	1/1/0/0	×	×
RemoteWMIWbemcomnDLLHijack	1/1/0/0	×	×
RemoteCreateFileSMB	0.99/0.99/0.5/0.5	×	×
WuaucIltCreateRemoteThread	1/1/0/0	×	×

Table 5.2: Decision trees on oversampled datasets. Data in each cell are "negative precision/negative recall/positive precision/positive recall", "×" indicates that methods cannot be used.

Table 5.2 displays the negative precision, negative recall, positive precision, and positive recall in each cell. The crosses in certain cells indicate that the oversampling methods cannot be applied to the corresponding dataset. As described in section 4.3, the SMOTE and ADASYN oversampling methods implement the K-Nearest Neighbor principle, which requires the number of samples to be more than the number of neighbors. Unfortunately, the majority of our datasets do not meet this requirement, making oversampling impossible.

For the datasets that can be oversampled, most of them did not yield satisfactory results. Take the "Remote WMI ActiveScriptEventConsumers" dataset as an example; as shown in the Figure 5.2, its confusion matrices generated through ROS, SMOTE, and ADASYN techniques indicate that SMOTE and ADASYN have predicted more positive data, leading to higher false positive rates than ROS. This increase in false results may indicate that the oversampling methods mistaken outsiders as minority samples, like e in Figure 4.3, leading to inaccuracies in the classification process. Furthermore, the few similarities between positive samples and the low number of positive samples contribute to the ineffectiveness of these oversampling techniques.

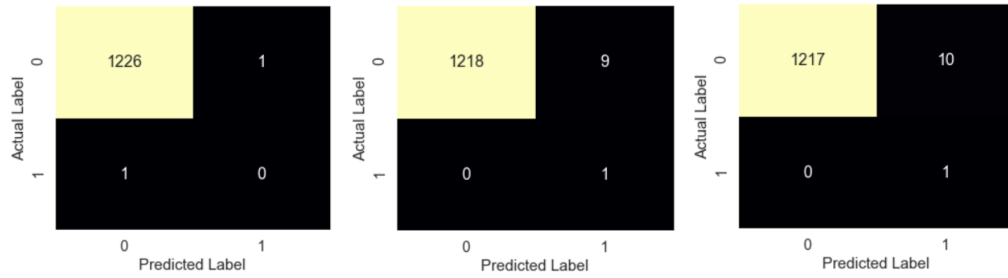


Figure 5.2: Confusion Matrixes of ROS, SMOTE and ADASYN on datasets for "Remote WMI ActiveScriptEventConsumers" from left to right

The only datasets that resulted in better outcomes than Scenario 1 are those of "PowerShell Remote Session" and "Domain DPAPI Backup Key Extraction," with positive precision scores of 0.67 and 1, respectively. However, only the latter dataset's results were satisfactory. Most of the other datasets yielded the same or worse results than Scenario 1, implying that our datasets do not support direct application of oversampling techniques as a remedy for the imbalanced data problem. Other methods are required to build an ideal IDS for such datasets.

Overall, the findings in Scenario 2 suggest that oversampling techniques have limited efficacy for improving IDS performance on imbalanced datasets, and other data handling techniques are needed for accurate intrusion detection.

5.4 Scenario 3

Scenario 3 is similar to Scenario 2, with the only difference being the model used for classification. In Scenario 2, decision trees showed poor performance on resampled data, which might be due to limitations with the decision tree algorithm's ability to handle complex relationships in data. To address this, we chose to use DNNs as classifiers for oversampled data. DNNs are capable of handling complex structures due to their complex network structures, making them ideal for datasets with complicated relationships.

In contrast to decision trees, DNNs are more sensitive to the data structure and require normalization of input data. As we discussed in section 4.2.2, various normalization methods need to be tested and compared to achieve the most optimal results. Through normalization and the utilization of DNNs, we aim to improve the classification performance for imbalanced datasets. And if the classification performance remains unsatisfactory despite implementing normalization and using DNNs as the classification model, we can conclude that oversampling techniques are not appropriate for our imbalanced datasets.

Upon performing a comparative analysis of various normalization techniques discussed in section 4.2.2, we observed that the standard scaling method, also known as z-score scaling, outperformed the other methods in terms of DNN model performance. The principle of standard scaling is demonstrated in equation 4.1.

To thoroughly evaluate the effectiveness of the oversampling method, we followed a series of preprocessing steps on 5 datasets with sufficient positive samples to leverage the SMOTE algorithm, as per Scenario 2. First, we oversampled the data using the SMOTE algorithm, followed by preprocessing the resampled data using the standard scaling technique. Next, we trained multiple deep neural network (DNN) models with varying numbers of hidden layers (ranging from 1-5) using the preprocessed data. After these procedures, we obtained the results presented in table 5.3.

Hunting Program	1 layer	2 layers	3 layers	4 layers	5 layers
LocalPwshExecution	1/1/0/0	1/0.96/0/0	1/1/0/0	1/0.83/0/0	1/1/0/0
RemotePwshExecution	1/0.71/0.01/1	1/1/0/0	1/1/0/0	1/1/0/0	1/0.97/0/0
PwshAlternateHosts	1/0.44/0/0	1/0.6/0/0	1/0.57/0/0	1/0.64/0/0	1/0.77/0/0
RemoteSCMHandle	1/0.88/0/0	1/0.88/0/0	1/0.88/0/0	1/0.88/0/0	1/0.88/0/0
MicrophoneDvcAccess	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0
RemoteWMIActiveScriptEventConsumers	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0

Table 5.3: SMOTE results of datasets using DNN classifier with different numbers of hidden layers. Data in each cell are "negative precision/negative recall/positive precision/positive recall".

It's imperative to note that our research is distinct from the study conducted by McElwee et al. [22] as we worked with datasets consisting of just two classes (normal and anomaly) and had between 2-13 features, as opposed to their datasets comprising of 6 classes with a large number of columns. As a result, it was not necessary to use a complex deep neural network structure in our experimentation. On applying the DNN classifier to the five datasets, we observed that increasing the number of hidden layers from 1 to 5 didn't have a significant impact on the classification results, except for some fluctuations arising from the randomness of data splitting. In most cases, using 2-3 hidden layers was sufficient to learn all the necessary information.

The results obtained in table 5.3 indicate that using SMOTE with the DNN model did not yield good performance. When compared with decision tree algorithm, the DNN showed almost identical performance, sometimes even with higher false positive rates due to more positive sample predictions. Unfortunately, none of the datasets in this scenario met our expectations, implying that oversampling techniques such as SMOTE are not suitable for our case. Furthermore, it proves that the undesirable results we obtained are not due to the decision tree algorithm, instead here the

problem lies with the resampled data itself.

5.5 Scenario 4

In this scenario, we employed CTGAN, a synthetic data generation methodology, to expand our dataset, as mentioned in section 4.4. Unlike the previous resampling techniques, CTGAN generates synthetic data and combines it with the authentic data to create training sets, selecting genuine data as testing sets. By generating synthetic data, we have more positive samples that can be put in the testing sets, ultimately resolving the predicament of limited positive samples that couldn't be used in either training or testing sets. Additionally, compared to oversampling techniques, generating synthetic data has the standout advantage that there are no restrictions on the relationships between the numbers of samples and their neighbors, allowing us to handle all the datasets in this scenario.

We applied CTGAN to all datasets, except for the two that yielded satisfactory results in Scenario 1. The basic processes involved in this scenario are shown in the fourth line of Figure 4.1, which includes using CTGAN to produce 1 million synthetic data, building balanced synthetic datasets by selecting samples from synthetic and authentic data, encoding all data in the datasets using label encoding, and finally classifying using the decision tree algorithm. To ensure the balanced synthetic datasets, we filtered synthetic data using analytics in the Threat Hunter Playbook and then downsampled synthetic positive or authentic negative samples, depending on the relationship between their numbers. By conducting these processes, we obtained the following results:

Hunting Program	Precision(-)	Recall(-)	Precision(+)	Recall(+)
LSASSMemoryReadAccess	1	1	1	1
DLLProcessInjectionCreateRemoteThread	1	1	1	1
ADObjectAccessReplication	1	1	1	0.75
ADModDirectoryReplication	1	1	1	1
LocalPwshExecution	1	1	0.67	1
RemotePwshExecution	1	1	0.57	0.8
PwshAlternateHosts	0.99	1	0.67	0.29
DomainDPAPIBackupKeyExtraction	1	1	1	1
RemoteWMIExecution	1	1	0.67	1
WMIEventing	1	1	1	1
WMIModuleLoad	1	1	1	0.8
LocalServiceInstallation	1	1	1	1
RemoteServiceInstallation	1	1	1	0.5
RemoteSCMHandle	1	1	1	0.75
RemoteInteractiveTaskMgrLsassDump	1	1	1	1
MicrophoneDvcAccess	1	1	0.62	1
RemoteWMIActiveScriptEventConsumers	1	1	0.5	0.57
RemoteWMIWbemcomnDLLHijack	1	1	1	0.67
RemoteCreateFileSMB	0.99	0.99	0.6	0.75
WuacltCreateRemoteThread	1	1	0.75	0.75

Table 5.4: Performance metrics of decision trees on datasets with synthetic data: Negative Precision, Negative Recall, Positive Precision, and Positive Recall

In this scenario, we successfully trained CTGAN models and generated 1 million synthetic data for 20 datasets. Out of these datasets, 7 provided perfect performance for classification, whereas the remaining ones performed significantly better than applying decision trees on original and oversampled data. Hence, we can conclude that using CTGAN to handle imbalanced problems works in our datasets.

We believe that the performance of the datasets can be further improved if we generate more synthetic data or use other methods to create new datasets instead of merely selecting samples randomly. For instance, for datasets such as "Remote SCM Handle", we filtered the 1 million samples using analytics in the Threat Hunter Playbook but did not obtain any results from the fifth analytics. Hence, this may lead to a lower recall as there were no training samples for that analytical event, and thus it cannot discover anomalies matching that analytic. To resolve this issue, we can try generating more synthetic samples or replicating specific authentic data samples that fit the CTGAN model. However, to avoid excessive computation and guarantee that the CTGAN models learn the authentic distribution in our datasets, we decided not to generate more synthetic samples in this research. Overall, we can confidently state that most of our datasets obtained good classification results in this scenario, and some almost achieve perfect precision and recall.

5.6 Scenario 5

In this scenario, we aimed to investigate whether DNNs can help in learning more intricate information. Similar to Scenario 3, we attempted to use the DNN model instead of the decision tree algorithm, as it could identify more complicated data structures with its multiple layers network. However, since some datasets in the last scenario already exhibited good performance, we only performed DNN classifiers on those datasets that didn't achieve satisfactory results in Scenario 4.

Furthermore, as mentioned before in section 5.4, we utilize DNNs with 1-5 hidden layers and ensure that our structure is sophisticated enough to learn all the information. Additionally, we need to perform normalization before inputting data into DNN models, and we continue to use standard scaling for normalization, as we observed that it gave the best results to DNN models previously.

After applying DNN models on various datasets, we arrived at the following results:

Hunting Program	1 layer	2 layers	3 layers	4 layers	5 layers
ADObjectAccessReplication	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0
LocalPwshExecution	0.98/0.88/0.01/0.1	0.98/0.69/0/0.1	0.98/0.49/0.01/0.2	0.98/0.48/0.01/0.2	0.98/0.43/0.01/0.3
RemotePwshExecution	0.99/0.75/0.02/0.44	0.99/0.84/0/0	0.99/0.74/0.02/0.44	0.99/0.72/0.02/0.44	0.99/1/0/0
PwshAlternateHosts	0.98/0.27/0/0.14	1/0.09/0.01/1	0/0/0.01/1	0.98/0.3/0/0.14	0.99/0.53/0/0.14
RemoteWMIExecution	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0
WMIModuleLoad	1/0.97/0.09/1	1/0.48/0/1	1/0.91/0.02/0.8	0/0/0/1	1/0.94/0.03/0.8
RemoteServiceInstallation	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0	1/1/0/0
RemoteSCMHandle	0/0/0.01/1	1/0.51/0.01/0.75	1/0.52/0.02/0.92	1/0.51/0.01/0.83	1/0.49/0.01/0.83
MicrophoneDvcAccess	1/0.57/0.01/0.5	1/0.64/0.01/0.5	1/0.58/0.01/1	0.99/0.69/0/0	1/0.66/0.01/0.5
RemoteWMIActiveScriptEventConsumers	1/0.01/0.01/1	0.99/0.93/0.03/0.3	1/0.02/0.01/1	1/0.01/0.01/1	1/0.02/0.01/1
RemoteWMIWbemcomnDLLHijack	1/0.63/0/0.33	1/0.58/0/0.67	1/0.26/0/1	1/0.66/0/0.33	1/1/0/0
RemoteCreateFileSMB	0/0/0.02/1	0/0/0.02/1	0/0/0.02/1	0/0/0.02/1	0/0/0.02/1
WuaucitCreateRemoteThread	1/0/0.01/1	0.99/0.22/0.01/0.75	0.67/0.01/0/0.5	0.33/0/0/0.5	0.99/0.41/0.01/0.75

Table 5.5: Synthetic datasets using DNN classifier with different numbers of hidden layers. Data in each cell are "negative precision/negative recall/positive precision/positive recall".

According to the results shown in 5.5, we can find that DNN models underperform in our binary classification problem compared to standard decision tree algorithms. It may happen because

of the insufficiency of training data since DNN models are deep and complex models, and they require large amounts of data to train effectively. If the training data is insufficient, the DNN may not have learned the underlying patterns and relationships in the data as well as a decision tree did. Apart from that, the results of DNN are unstable in our cases, which also shows that DNN is not a good choice for our datasets. Based on the outcomes presented in 5.5, it is apparent that DNN models deliver worse results in our binary-classification issue in contrast to standard decision tree algorithms. This underperformance might have occurred due to the insufficient training data since DNN models are deep and complex models, and they require vast amounts of data to train effectively. Inadequate data may lead to incomplete learning of underlying patterns and relationships in the data by the DNN compared to a decision tree.

Furthermore, the DNN results were unstable in our datasets, suggesting that DNN is not a suitable choice. In other words, the DNN model exhibits significant variability in prediction accuracy when trained multiple times on the same data. This instability impedes decision making and could have severe ramifications, making it unsuitable for real-world implementation in this scenario. Therefore, the application of decision tree algorithms in our datasets proves to be a more promising and reliable method for effective anomaly detection.

Chapter 6

Conclusion, Discussion and Future Work

The final chapter of our study proffers crucial insights obtained from our results. By comparing and analyzing our findings in detail while positing some potential hypotheses about the results, we will provide further explanations of our research design while engaging in a comprehensive discussion of the results. In this final section, we will also deliberate on any existing limitations in the study, propose potential improvements, and sketch out the trajectory for future research directions.

6.1 Conclusion

In Scenario 1, as depicted in Table 5.1, most of our datasets labelled using analytics in the Threat Hunter Playbook were biased towards the majority class (negative samples), leading to poor performance in the precisions and recalls of positive data. Additionally, we failed to apply decision trees for 4 datasets, as there were no positive samples left in the testing sets after randomly splitting the training and testing sets. Out of all the results we obtained, only 2 datasets, "SysKey Registry Keys Access" and "Registry Modification for Extended NetNTLM Downgrade" achieved satisfactory results of 1, 1 and 0.91, 1 for positive precision and positive recall, respectively. While 9 datasets got 0 for both their positive precision and recall. These results suggest that directly applying classification algorithms to the original datasets will not provide ideal performance for intrusion detection.

In Scenario 2 and 3, we utilized oversampling to deal with imbalanced datasets, with ROS, SMOTE, and ADASYN techniques. However, Table 5.2 indicates that 11 datasets could not use SMOTE and ADASYN since the number of samples was less than the minimum number of neighbours needed. Furthermore, for those datasets that employed SMOTE and ADASYN, performances remained dismal. With the exception of one dataset for "Local PowerShell Execution", which achieved a positive precision of 0.67 and a positive recall of 1 when using decision trees, all other datasets achieved less than 0.5 for either positive precision, positive recall, or both. Results in Scenario 3 showed that when using DNNs, performances got even worse. We constructed DNN models with 1-5 hidden layers, utilizing SMOTE to oversample data, but results in Table 5.3 demonstrated that DNNs mostly obtained 0 for positive precision and recall.

Given that oversampling techniques didn't work well for our datasets, we resorted to generating synthetic datasets using CTGAN, applying decision trees and DNNs in Scenario 4 and 5. Surprisingly, decision trees provided much better classification results on synthetic datasets, with 7 datasets obtaining perfect results, whereas the remaining ones achieved more than or equal to

0.5 for precisions and recalls. However, we attempted DNN models with 1-5 hidden layers for those datasets that didn't obtain optimal results in Scenario 4, and realized that DNNs provided significantly worse results than decision trees for each dataset. These findings suggest that synthetic data generated through CTGAN enables decision tree algorithms to achieve better overall performance compared to oversampling and direct classification methods.

6.2 Discussion

Our study accomplished the 3 objectives outlined in Section 1.3 after conducting various processes in 5 scenarios. From our analysis, we arrived at several conclusions concerning the datasets, machine learning techniques, and imbalanced data-handling methods.

6.2.1 Insufficient data in Threat Hunter Playbook and Security Datasets for building strong SIDS

Firstly, Chapter 3 involves a detailed investigation of the Threat Hunter Playbook and Security Datasets. It showed that there are a large amount of data missing in the datasets (in Appendix E), and labelled results in table 3.1 revealed that most datasets have extremely few positive samples. Additionally, the results obtained in Scenario 1 demonstrated that the Threat Hunter Playbook and its datasets cannot provide adequate information for directly building a SIDS. Consequently, to establish a more resilient research foundation for SIDS, it is necessary to gather additional data from Windows logs, particularly anomaly data, and include the missing data.

Our research indicated that developing a SIDS system solely based on analytics from the Threat Hunter Playbook's limited datasets could result in inaccurate and biased results, particularly concerning positive samples. Although by handling data appropriately, we can obtain better performance. With achieving the first objective, our study highlights the importance of thorough investigation and proper handling of data when utilizing the Threat Hunter Playbook and Security Datasets for building SIDS. Despite the challenges posed by missing data and imbalanced data-handling, we believe that with appropriate handling, the playbook and the datasets still hold value for SIDS-related research.

6.2.2 Decision Tree performs better than DNN in SIDS binary classification

When incorporating machine learning techniques, which are decision tree algorithms and DNNs in our research, we found that decision trees work well for building a SIDS in Scenario 4. By comparing Scenarios 2 and 3, 4 and 5, we conclude that decision tree algorithms are more effective in classifying binary anomaly datasets in contrast to the DNN models tested in our study. When we only have datasets with inadequate sample sizes, a model with complicated structures like DNNs is tend to give lower accuracy and be unstable. Consequently, structures with more hidden layers cannot help a lot in this situation. Therefore, when trying to incorporate machine learning methods into IDS, the sample sizes of datasets are always a significant feature that needed to be considered. More complicated structures can lead to worse performance sometimes when we don't have adequate information. Additionally, when building binary classification for SIDS like our cases, decision trees are worthy to be tried, since they can work well if used appropriately, which can save a large amount of time and effort than simple pattern matching.

With achieving the second objective, our study demonstrates the efficacy of decision tree algorithms for classification in a binary anomaly detection scenario. This finding suggests that

decision trees are useful tools to include in future research. Additionally, our study highlights the need to consider the sample sizes of datasets when incorporating machine learning methods into IDS. This consideration can enable researchers to avoid obtaining models with unstable and low accuracy results due to insufficient data.

6.2.3 CTGAN as an Effective Solution for Handling Imbalanced Datasets in SIDS

Lastly, results in our study's best scenario, Scenario 4, show that generating synthetic data using CTGAN can be a good way for dealing with extremely imbalanced datasets. Oversampling techniques like ROS, SMOTE and ADASYN are mostly common methods to deal with imbalanced data, however, these traditional methods cannot work well in extreme cases since they have the minimum requirement for the numbers of samples to generate neighbors. In contrast, synthetic data generation via CTGAN does not have any requirement, it works even when there is only one sample in the minority class. According to our results in Scenario 4, we can say that CTGAN provides a more reliable approach to tackling class imbalance and enhancing the classification performance for SIDS.

With achieving the third objective, our study's results highlight the usefulness of CTGAN for generating synthetic data to address the challenges of class imbalance in SIDS. This finding implies that SIDS developers should consider CTGAN when their datasets have extreme class imbalance to avoid obtaining inaccurate and biased results.

Overall, our study presents a practical and feasible methodology for building SIDS based on Threat Hunter Playbook and Security Datasets, and highlights how integrating appropriate machine learning techniques can make the detection system more effective and efficient. Synthetic data generation, as exemplified by CTGAN in our study, can enable IDS developers to address class imbalance issues effectively. Moreover, our study's novel findings can be expanded beyond the domain of intrusion detection or cybersecurity. Specifically, we highlight the efficacy of synthetic data generation in overcoming the challenges of extremely imbalanced datasets in all situations.

6.3 Future Work

Despite our study's valuable findings, there exist several limitations in our experiments that require further investigation. Although our best scenario demonstrated a significant improvement in positive precision and recall for some datasets, they were still far from perfect. To enhance the results, we propose several potential ways to improve the methodology employed in our study.

One possible approach is to make the synthetic datasets larger to counter the insufficient samples in some analytics, which can be achieved by generating more than 1 million samples using CTGAN. Additionally, instead of randomly selecting samples from all generated data, alternative downsampling techniques can retain more informative ones and be employed to produce better-performing synthetic datasets. Moreover, our study primarily focused on CTGAN as the synthetic data generation method. However, other approaches, such as Deep Belief Networks (DBNs), could be experimented with and compared to CTGAN for their efficacy in resolving class imbalance issues.

In addition to the generation of synthetic datasets, data preprocessing also can be enhanced. One potential method to explore is the use of one-hot encoding instead of label encoding for

categorical variables. Since categorical variables do not have any inherent order or hierarchy in our datasets, using one-hot encoding could provide a better representation of the data and enhance the classification performance of SIDS. Additionally, we can investigate the benefits of combining normalization with batch normalization for DNN models. This technique can enable the model to learn better representations of the data and it could lead to improved accuracy and stability.

Moreover, further research is crucial to extend this work and explore the application of similar problems. One such area deserving attention is multi-class anomaly detection models with more complex data structures. Our analysis in this study was limited to binary classification models, and we did not explore multi-class models. Besides, we did not adequately explore the design of DNN architectures or hyperparameter tuning, which could have negatively affected the performance of these models. Therefore, future researchers should consider these factors when developing DNN models for SIDS. Additionally, alternative methods such as support vector machines (SVM) could also contribute to the development of IDS.

Overall, while our study provides valuable insights on building SIDS based on the Threat Hunter Playbook and Security Datasets using machine learning techniques, we recognize the need for further research and exploration of alternative algorithms and techniques. This continued development can bring us closer to building an accurate and reliable SIDS capable of dealing with the dynamic and evolving threats in real-world scenarios.

Appendix A

User Manual

A.1 Running on HPC

Due to the high computing ability and storage requirement of this project, we used the High-Performance Computing (HPC) facility provided by the University of Aberdeen to run part of the project. However, if you have devices with sufficiently large memory and computing ability, you may not need to use HPC for the project. Nonetheless, we recommend using HPC for time-saving purposes. The steps to run Jupyter Notebook remotely on the University of Aberdeen's server, Maxwell, using your own devices are outlined below. The process of running Jupyter Notebook on other HPCs may vary slightly but should be similar.

A.1.1 Register an HPC account

To begin using HPC resources, the first step is to register for a personal account. To request an account, you can submit the Request Form for Maxwell on the school's website (<https://www.abdn.ac.uk/research/digital-research/hpc-for-research-1097.php>). Alternatively, you can contact the digital research team by email (digitalresearch@abdn.ac.uk). However, before proceeding with the account request, it is essential to ensure that you have read and understood the user responsibilities and conditions for using Information Technology facilities, which are posted by the University of Aberdeen.

A.1.2 Install and Run F5 Network VPN

This step is necessary only if you aim to access University resources remotely using personal devices. If you use a University device, you may skip this step. You can install the VPN from the University website (<https://www.abdn.ac.uk/toolkit/systems/remote-access/>) and access University resources by selecting the desired service (<https://remote.abdn.ac.uk>) off-campus. Once f5 VPN is installed, for instance, you can select the maxlogin1 application, and its status will be displayed as follows:

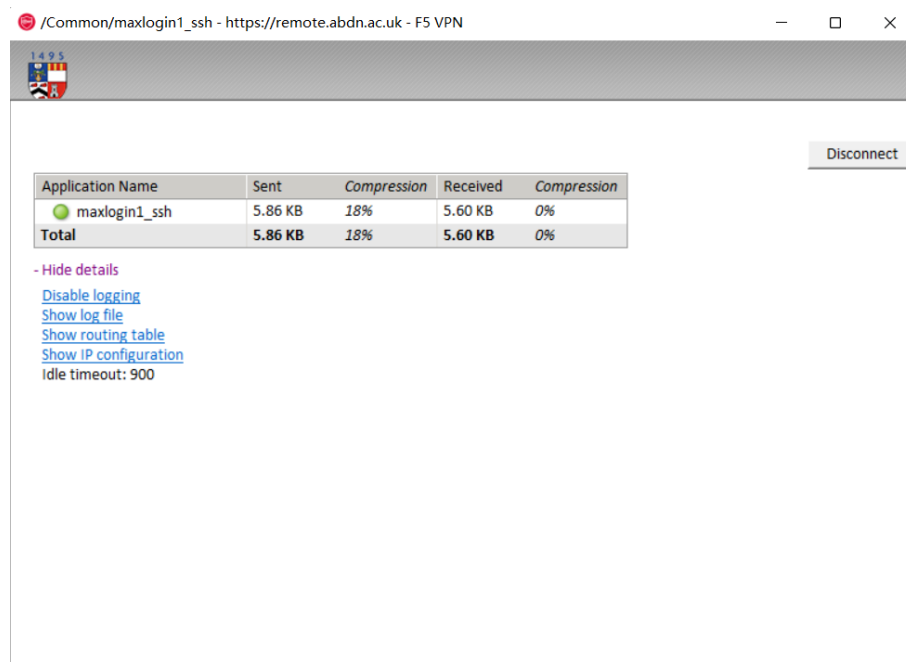


Figure A.1: Access maxlogin1 via f5 networks VPN

A.1.3 Install and Run PuTTY

PuTTY is a free and open-source terminal emulator, serial console, and network file transfer application that supports various network protocols, including SSH. To use it, you must first download the installer from the official website (<https://www.putty.org/>) that is compatible with your system. Next, choose a pathway for installation and customize the installation setting based on your preference. Once installed successfully, start the PuTTY software, and you will see the configuration as shown below:

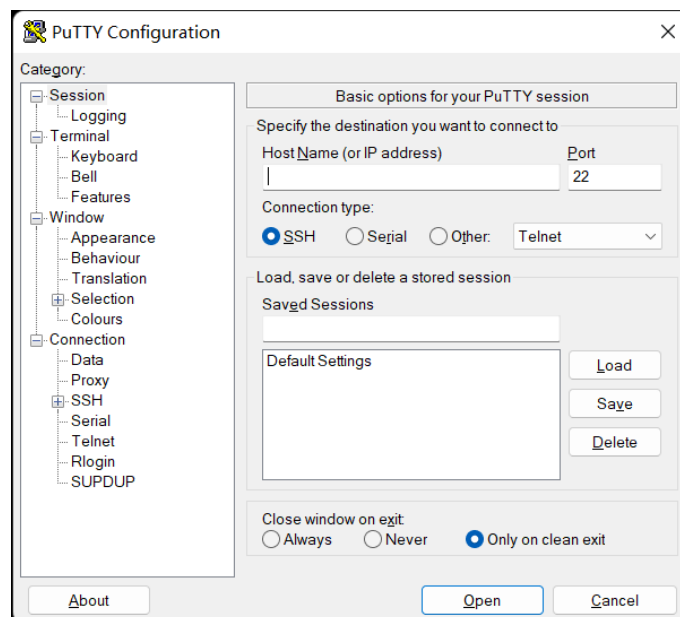


Figure A.2: PuTTY configuration

To access the desired Host Name, you should enter the format "username@address," for

```

u15yl21@maxlogin1:~
Using username "u15yl21".
Keyboard-interactive authentication prompts from server:
Password:
End of keyboard-interactive prompts from server
Last failed login: Wed Apr 26 14:42:30 BST 2023 from 139.133.8.131 on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Sat Apr 22 16:37:41 2023 from 139.133.8.131
SLURM: Your account, u15yl21, is ready to submit Slurm jobs.
Info: Loaded slurm into the modular environment.
Info: Loaded lmod into the modular environment.

(base) [u15yl21@maxlogin1(maxwell) ~]$

```

Figure A.3: PuTTY successfully logged in

instance, "u15yl21@maxlogin1.abdn.ac.uk." Select "SSH" as the Connection Type and save your session to connect conveniently in the future. Once you activate the session, the system prompts you to input your password. Only if your password matches the Host Name you provided, can you connect to the server. Upon successful connection, your terminal should look like the following:

A.1.4 Request for Allocated Resources

To leverage the high-speed computing capability of the HPC, you must request computing resources using the SLURM scheduler command "salloc." For example, you can request 4 CPUs per task and 8 gigabytes memory per CPU using the following command:

```
salloc --cpus-per-task 4 --mem-per-cpu 8G --partition spot-compute
```

Once you have obtained your requested resources, your terminal should display information similar to the following:

```

(base) [u15yl21@maxlogin1(maxwell) ~]$ salloc --cpus-per-task 4 --mem-per-cpu 8G
--partition spot-compute
salloc: Pending job allocation 1135437
salloc: job 1135437 queued and waiting for resources
salloc: job 1135437 has been allocated resources
salloc: Granted job allocation 1135437
salloc: Waiting for resource configuration
salloc: Nodes compute006 are ready for job
SLURM: Your account, u15yl21, is ready to submit Slurm jobs.
Info: Loaded slurm into the modular environment.
Info: Loaded lmod into the modular environment.

```

Figure A.4: Being allocated resources successfully

A.1.5 Prepare Anaconda Environment

When setting it up initially, you can use the following command to verify which versions of Anaconda are available on your server:

```
module spider anaconda
```

In the HPC server of the University of Aberdeen, anaconda3/2021.05 should be available. You can load the module by running the command:

```
module load anaconda3/2021.05
```

At first, you should install a Python environment based on your specific requirements. For example, you can install a Python version 3.7 environment by using the conda tool, as illustrated below:

```
conda create -n myJupyter_3.7 python=3.7 jupyter
```

And then, you can activate the environment you have:

```
conda activate myJupyter_3.7
```

Moreover, during the first-time setup, you may need to install a Python kernel to ensure the smooth running of Jupyter Notebook. For instance, you can run the command below to install it:

```
python -m ipykernel install --user --name myenv
```

A.1.6 Build an SSH Tunnel

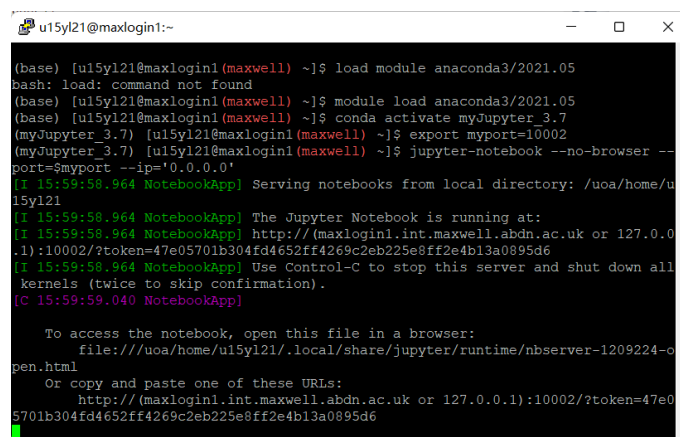
To run Jupyter Notebook via HPC on your personal devices, it is necessary to establish an SSH tunnel between your local machine and the server. On the server side, once you activate the environment, you need to configure the forwarding of the notebook data over an unused port, such as 10002, as shown below:

```
export myport=10002
```

Afterwards, you can commence the notebook with a specified port and IP address, such as by running the following command:

```
jupyter-notebook --no-browser --port=$myport --ip='0.0.0.0'
```

And if the notebook runs successfully, you should be notified of the URLs for connection, which should be something similar to the figure below:



```

u15yl21@maxlogin1:~
(base) [u15yl21@maxlogin1(maxwell) ~]$ load module anaconda3/2021.05
bash: load: command not found
(base) [u15yl21@maxlogin1(maxwell) ~]$ module load anaconda3/2021.05
(base) [u15yl21@maxlogin1(maxwell) ~]$ conda activate myJupyter_3.7
(myJupyter_3.7) [u15yl21@maxlogin1(maxwell) ~]$ export myport=10002
(myJupyter_3.7) [u15yl21@maxlogin1(maxwell) ~]$ jupyter-notebook --no-browser --port=$myport --ip='0.0.0.0'
[I 15:59:58.964 NotebookApp] Serving notebooks from local directory: /uoa/home/u15yl21
[I 15:59:58.964 NotebookApp] The Jupyter Notebook is running at:
[I 15:59:58.964 NotebookApp] http://(maxlogin1.int.maxwell.abdn.ac.uk or 127.0.0.1):10002/?token=47e05701b304fd4652ff4269c2eb225e8ff2e4b13a0895d6
[I 15:59:58.964 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:59:59.040 NotebookApp]

To access the notebook, open this file in a browser:
file:///uoa/home/u15yl21/.local/share/jupyter/runtime/nbserver-1209224-open.html
Or copy and paste one of these URLs:
http://(maxlogin1.int.maxwell.abdn.ac.uk or 127.0.0.1):10002/?token=47e05701b304fd4652ff4269c2eb225e8ff2e4b13a0895d6

```

Figure A.5: Run Jupyter Notebook Successfully

Next, on the client-side, typically on your personal computer or device, you must open a new terminal and initiate the SSH tunnel between the server (master node) and your local machine. The command required to execute this step will resemble the following:

```
ssh -NL 10002:maxlogin1.abdn.ac.uk:10002 u15yl21@maxlogin1.abdn.ac.uk
```

After initiating the SSH tunnel, you will be prompted for a password on the client-side. It is crucial to note that you will not receive any output if the connection is successful, and you must keep the terminal active. With the tunnel established, you may proceed to access your notebook by opening your browser and utilizing the URL displayed in Figure A.5.

A.2 Running Jupyter Notebook

Whether running the project via HPC or on your personal devices, you can execute this project quickly by running scripts in Jupyter Notebook. Jupyter Notebook is an interactive and responsive platform that allows users to combine a narrative, code, and computational results in one place. By utilizing code to perform analysis and models to present results in tables and graphs, users can clearly see the results that correspond to each input cell.

Before executing scripts, you must ensure that you have a suitable Python environment installed with all the necessary packages. To check for the required libraries, you can refer to the "import" section in each script. Generally, libraries such as numpy, pandas, Tensorflow, sklearn, seaborn, and sdv are necessary to run the program smoothly. You can install these libraries using the pip command in the code cell. For instance, to install TensorFlow, you can use the following command:

```
! pip install tensorflow
```

Additionally, it is critical to ensure that you utilize the correct pathways when interacting with files such as reading data files and saving generated data. To execute the code present in code cells, you may either select "Run Selected Cell" from the top menu bar or opt for the "Ctrl" + "Enter" keyboard shortcut. A typical Jupyter Notebook page is displayed in Figure A.6.

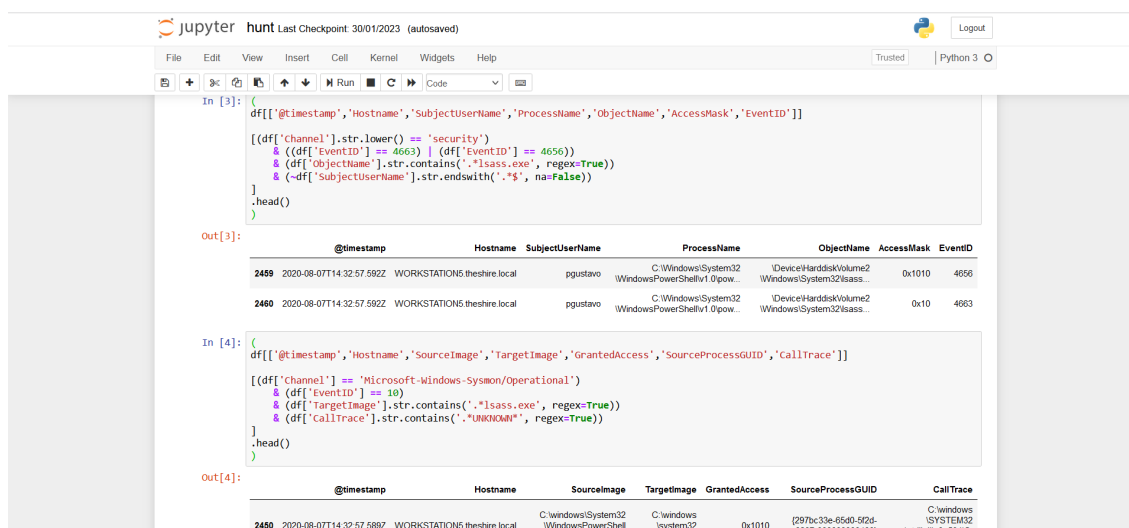


Figure A.6: Example of Jupyter Notebook

Appendix B

Maintenance Manual

B.1 Installation

The only required software for this project is Jupyter Notebook. To install Jupyter Notebook, it is highly recommended that you download a scientific Python distribution that includes scientific Python packages. The most commonly used distribution is Anaconda. To install Anaconda, start by downloading the Anaconda distribution, Python3, 64 bits, from the website: <https://www.anaconda.com/download/>. Then you can follow the instructions on the download page to install it using the default settings for a single user. To launch the notebook, click the Jupyter Notebook icon or run the command "jupyter notebook" on the command prompt.

Alternatively, if you already have an existing Python installation on your computer, you can use pip, Python's package manager, to install Jupyter Notebook. Start by upgrading your pip to the latest version to ensure compatibility, then use the pip command to perform the installation.

```
pip3 install --upgrade pip
pip3 install jupyter
```

B.2 Software Dependencies

All the libraries imported by the scripts are necessary, and some of them may install automatically. Below is a comprehensive list of all libraries utilized in our code and brief descriptions of their function in our program:

Dependency	Usage
os	For getting file paths
collections	For items counting and debugging
matplotlib	For graphs drawing
seaborn	For graphs drawing
numpy	For data handling
pandas	For data handling and file reading, writing
tensorflow	For machine learning (DNN)
sklearn	For preprocessing, sets splitting, machine learning (Decision Tree), results evaluation
sdv	For synthetic data generation (CTGAN)
imblearn	For data resampling

Table B.1: Software dependencies and their usage in the program

To install the dependencies above, you can simply use the pip command in Python.

B.3 Running the System

Although it is ideal to follow the sequence of the 5 scenarios to complete the entire project, it is possible to work on each part of the project independently as we have provided all the necessary datasets. The project comprises 6 primary functions, which can be executed by running the following programs:

- To run programs in Threat Hunter Playbook, for specific datasets, you can run the hunt.ipynb under the names of datasets, under the folder /hunt program/.
- To run Scenario 1, which should be original data with decision tree classifiers, you can run /Resampled Data/Scenario 1.ipynb. You should make sure that you already have the datasets under /Resampled Data/.
- To run Scenario 3, which should be oversampled data with DNN classifiers, you can run /Resampled Data/Scenario 3.ipynb. You should make sure that you already have the datasets under /Resampled Data/.
- To run Scenario 4, which should be synthetic data with decision tree classifiers, you can run all the ipynb files with CTGAN as prefixes under /Synthetic Data/. You should make sure that you already have the datasets under /Synthetic Data/datasets/.
- To run Scenario 5, which should be synthetic data with decision tree classifiers, you can run /Synthetic Data/Scenario 5.ipynb. You should make sure that you already have the generated datasets under /Synthetic Data/generated datasets/.

B.4 Requirements

When fitting data to CTGAN models and generating large volumes of synthetic data with the CTGAN algorithm, processing large datasets with multiple features can consume significant space and memory. Therefore, running programs via HPC is recommended. The exact memory and space requirements will differ based on the size of the dataset. We have observed that memory errors occur when processing datasets with 10 columns and 3000 rows on a device with 7.79G RAM available. However, utilizing HPC with 8 CPUs allocated for each task and 16G memory allocated for each CPU is capable of handling these datasets, although some waiting time may still be necessary to achieve the desired results.

B.5 Key File Paths

These are the file paths for some of the important files used in the project:

1. Hunting Programs from Threat Hunter Playbook:
/hunt program/
2. Scenario 1:
/Resampled Data/Scenario 1.ipynb

3. Scenario 2:
/Resampled Data/Scenario 2.ipynb
4. Scenario 3:
/Resampled Data/Scenario 3.ipynb
5. Scenario 4:
/Synthetic Data/CTGAN_ADModDirectoryReplication.ipynb
/Synthetic Data/CTGAN_ADOBJECTAccessReplication.ipynb
/Synthetic Data/CTGAN_...
6. Scenario 5:
/Synthetic Data/Scenario 5.ipynb
7. Reading Datasets:
/Synthetic Data/datasets/
Or, /Resampled Data/
8. Saving, reading synthetic data:
/Synthetic Data/generated_datasets/
9. Saving, loading synthesizer models:
/Synthetic Data/models/
10. Saving decision tree graphs:
/Synthetic Data/DecisionTree/

B.6 Directions for Future Improvements

There are numerous possible enhancements that may be required for our project in the future.

B.6.1 Datasets

Based on our results, it is evident that the Security Datasets we utilized did not provide sufficient information for classification, and therefore, the most significant approach to enhance the performance of SIDSs is to extend our investigation to other trustworthy datasets with a higher number of anomaly samples.

B.6.2 Preprocessing Data

Regarding data preprocessing, we currently employ encoding and normalization techniques. To improve the systems, alternative encoding methods like one-hot encoding can be utilized in place of label encoding. Furthermore, for DNN models, other normalization methods such as batch normalization techniques may be considered.

B.6.3 Synthetic Data

Given additional time and resources, we could generate over 1 million samples by using CTGAN to synthesize data. Additionally, other downsampling techniques can be explored to maintain the actual distribution of the datasets, and other synthetic data generation methods such as Deep Belief Networks (DBNs) can be employed.

B.6.4 DNN Classification

As the DNN models in our research did not produce satisfactory results, other architectures and hyperparameters pruning of DNN models should be explored.

B.6.5 Others

Apart from the improvements mentioned above, other machine learning algorithms except for decision trees and DNN can also be investigated to give a better performance of the SIDSs.

B.7 Bug Reports

Initially, this project was run on our personal computer, which had Python 3.7.3 installed on the Windows operating system, and the file paths were set up accordingly. We used SDV 0.18.0 in this setup. However, some of the datasets in Scenario 4 were run on HPCs with Linux operating systems, and thus the file paths were modified to comply with Linux rules, and we used SDV 1.0. Moreover, starting from March 28, 2023, the older version of SDV is no longer supported or maintained. Therefore, we switched to the new version of SDV, which has a different syntax. If you wish to run the code on HPC instead of Windows, you need to change the syntax as shown in the following examples.

For using CTGAN, you should change the code about SDV from:

```
from sdv.tabular import CTGAN

model = CTGAN(cuda=True)
model.fit(dataframe)
# save the model
model.save('model.pkl')
# load the model
model = CTGAN.load('models.pkl')

# generate synthetic data
new_data = model.sample(num_rows=1000000)
```

to:

```
from sdv.metadata import SingleTableMetadata
from sdv.single_table import CTGANSynthesizer

metadata = SingleTableMetadata()
metadata.detect_from_dataframe(data=dataframe)
synthesizer = CTGANSynthesizer(metadata, cuda=True)
synthesizer.fit(dataframe)
# save the model
synthesizer.save(filepath='models.pkl')
# load the model
synthesizer = CTGANSynthesizer.load(filepath='model.pkl')
```

```
# generate synthetic data
new_data = synthesizer.sample(num_rows=1000000)
```

And for the code about the path of files, you should change the code from:

```
datasetJSONPath = os.getcwd() + "\\datasets\\datasets.json"
```

to:

```
datasetJSONPath = os.getcwd() + "/datasets/datasets.json"
```

Appendix C

Features Used in Datasets

Table C.1: Features used in datasets.

Hunt Program	Features
LSASSMemoryReadAccess	"Channel", "EventID", "ObjectName", "SubjectUserName", "TargetImage", "CallTrace", "ImageLoaded", "@timestamp", "ProcessGuid", "SourceProcessGUID"
DLLProcessInjectionCreateRemoteThread	"Channel", "EventID", "StartModule", "StartFunction", "TargetFilename", "ImageLoaded", "Hostname"
ADObjectAccessReplication	"Channel", "EventID", "AccessMask", "Properties", "SubjectUserName", "LogonType", "SubjectLogonId", "TargetLogonId"
ADModDirectoryReplication	"Channel", "EventID", "ObjectServer", AccessMask", "ObjectType", "AttributeLDAPDisplayName", "AttributeValue"
LocalPwshExecution	"Channel", "EventID", "NewProcessName", "ParentProcessName", "Image", "ParentImage", "ImageLoaded", "Description", "PipeName"
RemotePwshExecution	"Channel", "EventID", "Message", "DestPort", LayerRTID", "ParentProcessName", "NewProcessName", "ParentImage", "Image", "DestinationPort", "User"
PwshAlternateHosts	"Channel", "EventID", "Message", "Description", "ImageLoaded", "Image", "PipeName"
DomainDPAPIBackupKeyExtraction	"Channel", "EventID", "AccessMask", "ObjectName", "LogonType", "SubjectUserName", "Hostname", "ShareName", "RelativeTargetName", "SubjectLogonId", "TargetLogonId", "Message"
Continued on next page	

Table C.1 – continued from previous page

Hunt Program	Features
RegKeyAccessSyskey	"Channel", "EventID", "ObjectType", "ObjectName"
RemoteWMIExecution	"Channel", "EventID", "ParentProcessName", "TargetLogonId", "LogonType", "SubjectUserName", "ParentImage", "LogonId"
WMIEventing	"Channel", "EventID"
WMIModuleLoad	"Channel", "EventID", "ImageLoaded", "Image"
LocalServiceInstallation	"Channel", "EventID"
RemoteServiceInstallation	"Channel", "EventID", "SubjectUserName", "SubjectLogonId", "TargetLogonId", "LogonType"
RemoteSCMHandle	"Channel", "EventID", "ObjectType", "ObjectName", "AccessMask", "SubjectLogonId", "PrivilegeList", "Application", "LayerRTID", "Image", "LogonType", "SubjectUserName", "TargetLogonId"
RemoteInteractiveTaskMgrLsassDump	"Channel", "EventID", "Image", "TargetFilename", "SourceImage", "TargetImage", "CallTrace", "ProcessGuid", "SourceProcessGUID", "LogonId"
RegModExtendedNetNTLMDowngrade	"Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "SubjectLogonId", "ObjectValueName", "TargetObject"
MicrophoneDvcAccess	"Channel", "EventID", "ObjectType", "ObjectName", "ProcessName", "SubjectLogonId", "ObjectValueName", "TargetObject"
RemoteWMIActiveScriptEventConsumers	"Channel", "EventID", "Message", "Image", "NewProcessName", "ImageLoaded", "LogonType", "ProcessName", "ProcessGuid", "ProcessId"
RemoteDCOMIertUtilDLLHijack	"Channel", "EventID", "RelativeTargetName", "AccessMask", "SubjectUserName", "Image", "TargetFilename", "ImageLoaded"
RemoteWMIWbemcomnDLLHijack	"Channel", "EventID", "RelativeTargetName", "SubjectUserName", "AccessMask", "Image", "TargetFilename", "ImageLoaded"
Continued on next page	

Table C.1 – continued from previous page

Hunt Program	Features
RemoteCreateFileSMB	"Channel", "EventID", "ShareName", "SubjectUserName", "SubjectLogonId", "AccessMask", "Image", "RelativeTargetName", "TargetFilename"
WuaucLtCreateRemoteThread	"Channel", "EventID", "Image", "CommandLine", "Signed", "SourceImage", "ImageLoaded", "TargetFilename", "SourceProcessGuid", "ProcessGuid"

Appendix D

Datasets

Datasets used by hunting programs and their description are as follows:

- **LSASS Memory Read Access:** Empire Mimikatz LogonPasswords. This dataset represents adversaries reading credentials from the memory contents of lsass.exe. One popular tool performing this behavior is Mimikatz.
- **DLL Process Injection via CreateRemoteThread and LoadLibrary:** Empire Invoke DLLInjection. This dataset represents a threat actor injecting a Dll (On Disk) into an arbitrary process via LoadLibrary and executed by CreateRemoteThread APIs.
- **Active Directory Object Access via Replication Services:** Empire DCSync. This dataset represents adversaries abusing Active Directory Replication services to retrieve secret domain data (i.e. NTLM hashes) from domain accounts.
- **Active Directory Root Domain Modification for Replication Services:** Empire Powerview Add-DomainObjectAcl. These datasets represent adversaries with enough permissions (i.e. domain admin) to add an access control entry (ACE) to the discretionary access control list (DACL) of an Active Directory object (i.e Root Domain). One example could be adversaries modifying the root domain DACL to allow a specific domain user, despite being in no privileged groups and not having local admin rights on the domain controller itself, to use Active Directory replication services and obtain secret domain data (i.e. Other user NTLM Hashes)
- **Local PowerShell Execution:** Empire VBS Execution. This dataset represents adversaries executing a VBS script as a launcher for initial access.
- **PowerShell Remote Session:** Empire Invoke PSRemoting. This dataset represents adversaries executing malicious code on remote hosts using PowerShell Remoting (WinRM).
- **Alternate PowerShell Hosts:** Empire Invoke PSRemoting. This dataset represents adversaries executing malicious code on remote hosts using PowerShell Remoting (WinRM).
- **Domain DPAPI Backup Key Extraction:** Empire Mimikatz Backup Keys. This dataset represents adversaries retrieving the DPAPI Domain Backup Key from the DC via RPC LSARPC methods over SMB.

- **SysKey Registry Keys Access:** Empire Mimikatz SAM Extract Hashes. This dataset represents adversaries calculating the SysKey to decrypt Security Account Manager (SAM) database entries (from registry or hive) and get NTLM, and sometimes LM hashes of local accounts password.
- **WMI Win32_Process Class and Create Method for Remote Execution:** Empire Invoke WMI. This dataset represents an adversary remotely executing code via WMI. This dataset focuses on the use of the WMI Win32_Process class and method Create to execute code remotely.
- **WMI Eventing:** Empire Elevated WMI Eventing. This dataset represents adversaries leveraging WMI subscriptions locally for persistence.
- **WMI Module Load:** Empire PSInject. This dataset represents adversaries reflectively loading/intecting a portable executable (PE) (not on disk) into a process via WriteprocessMemory and executed via CreateRemoteThread APIs
- **Local Service Installation:** Empire Invoke PsExec. This dataset represents adversaries remotely creating and starting a service via RPC methods over TCP.
- **Remote Service creation:** Empire Invoke PsExec. This dataset represents adversaries remotely creating and starting a service via RPC methods over TCP.
- **Remote Service Control Manager Handle:** Empire Find Local Admin Access. This dataset represents adversaries using the OpenSCManagerW Win32API call to establish a handle to the remote host and verify if the current user context has local administrator access to the target.
- **Remote Interactive Task Manager LSASS Dump:** RDP TaskManager LSASS Dump. This dataset represents adversaries using RDP and task manager interactively and dump the memory space of lsass.
- **Registry Modification for Extended NetNTLM Downgrade:** Empire Invoke Internal-Monologue. This dataset represents adversaries downgrading the challenge/response authentication protocol used for network logons, the minimum security negotiated for applications using NTLMSSP, and security settings that restrict outgoing NTLM traffic to remote servers in an environment
- **Access to Microphone Device:** MSF Record Mic. This dataset represents adversaries accessing the microphone of an endpoint.
- **Remote WMI ActiveScriptEventConsumers:** Covenant Remote WMI Eventing ActiveScriptEventConsumers. This dataset represents adversaries using WMI event subscriptions (ActiveScriptEventConsumers) remotely to move laterally.
- **Remote DCOM IErtUtil DLL Hijack:** Covenant Remote DCOM Iertutil DLL Hijacking. This dataset represents adversaries abusing a DLL hijack vulnerability found in the execution of the DCOM InternetExplorer.Application class for lateral movement.

- **Remote WMI Wbemcomn DLL Hijack:** Covenant Remote WMI Wbemcomn DLL Hijacking. This dataset represents adversaries abusing a DLL hijack vulnerability found in the execution of the WMI provider host (wmiprvse.exe) for lateral movement.
- **SMB Create Remote File:** Covenant Remote File Copy. This dataset represents a threat actor remotely copying a file over SMB (CreateRequest).
- **Wuaucvt CreateRemoteThread Execution:** Covenant Wuaucvt CreateRemoteThread Execution. This dataset represents adversaries proxy executing code via the Windows Update client utility. In order to bypass rules looking for the binary reaching out directly to the Internet, this dataset shows the binary creating and running a thread in the virtual address space of another process via the CreateRemoteThread API.

Descriptions of datasets are given by Rodriguez et al. [31], more information such as simulation metadata can be found on the website (<https://securitydatasets.com/introduction.html>).

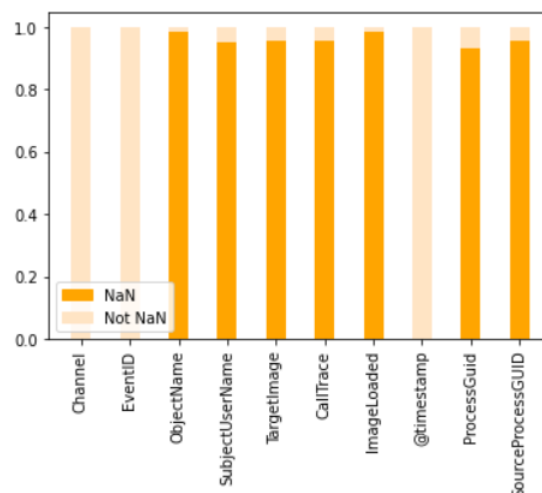
Appendix E

Data Missing in Datasets

To determine the amount of missing data present in each dataset, we count the number of NaN values for each feature. Below is a summary of the percentage of NaN values present for each feature in the datasets.

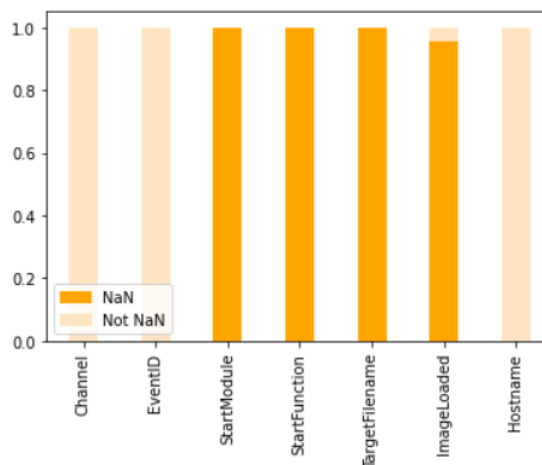
Percentages of missing data in Datasets for "LSASS Memory Read Access":

Channel:0/6026
EventID:0/6026
ObjectName:5933/6026
SubjectUserName:5740/6026
TargetImage:5754/6026
CallTrace:5754/6026
ImageLoaded:5945/6026
@timestamp:0/6026
ProcessGuid:5619/6026
SourceProcessGUID:5754/6026



Percentages of missing data in Datasets for "DLL Process Injection via CreateRemoteThread and LoadLibrary":

Channel:0/12200
EventID:0/12200
StartModule:12199/12200
StartFunction:12199/12200
TargetFilename:12165/12200
ImageLoaded:11630/12200
Hostname:0/12200



Percentages of missing data in Datasets for "Active Directory Object Access via Replication"

Services":

Channel:0/8465

EventID:0/8465

AccessMask:8371/8465

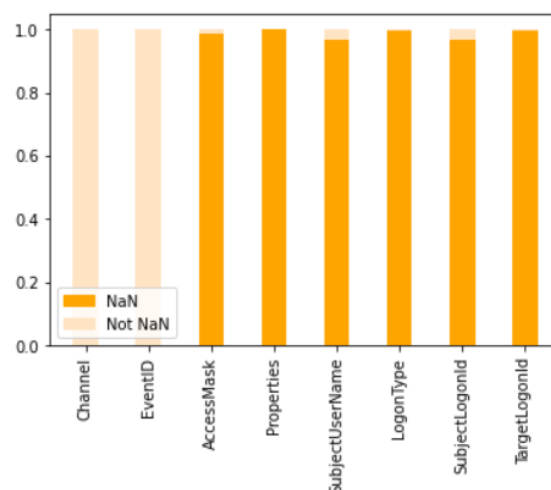
Properties:8462/8465

SubjectUserName:8193/8465

LogonType:8443/8465

SubjectLogonId:8193/8465

TargetLogonId:8428/8465



Percentages of missing data in Datasets for "Active Directory Root Domain Modification for Replication Services":

Channel:0/9002

EventID:0/9002

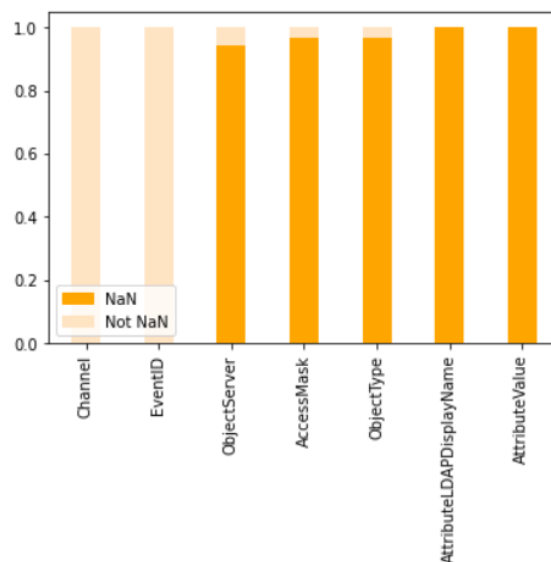
ObjectServer:8469/9002

AccessMask:8728/9002

ObjectType:8725/9002

AttributeLDAPDisplayName:8996/9002

AttributeValue:8996/9002



Percentages of missing data in Datasets for "Local PowerShell Execution":

Channel:0/2067

EventID:0/2067

NewProcessName:2062/2067

ParentProcessName:2062/2067

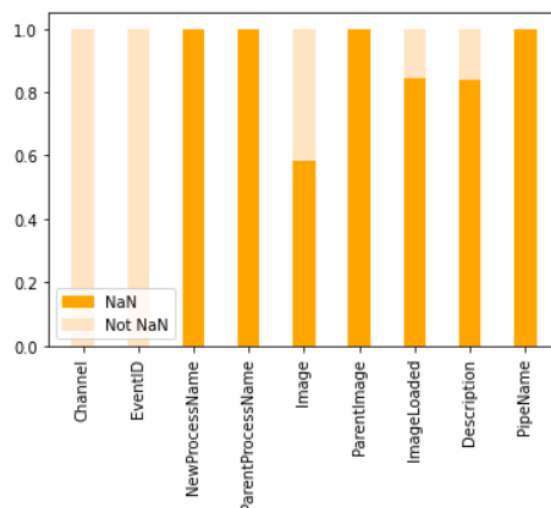
Image:1201/2067

ParentImage:2062/2067

ImageLoaded:1743/2067

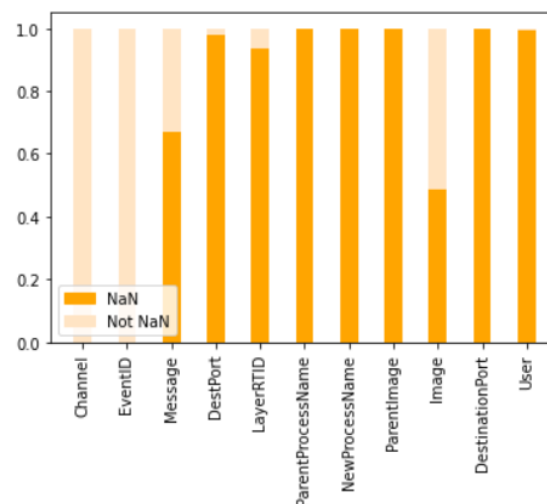
Description:1738/2067

PipeName:2060/2067



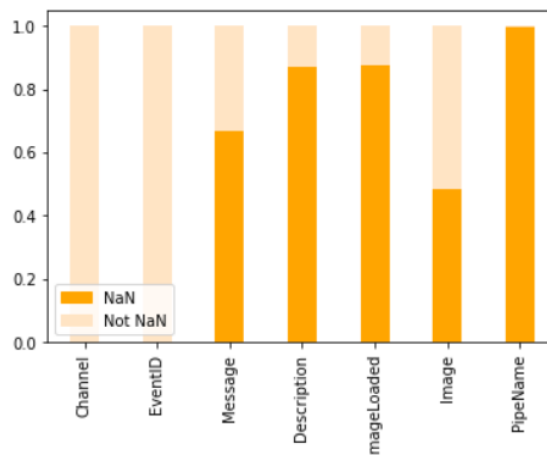
Percentages of missing data in Datasets for "PowerShell Remote Session":

Channel:0/2744
 EventID:0/2744
 Message:1835/2744
 DestPort:2681/2744
 LayerRTID:2572/2744
 ParentProcessName:2739/2744
 NewProcessName:2739/2744
 ParentImage:2739/2744
 Image:1332/2744
 DestinationPort:2734/2744
 User:2722/2744



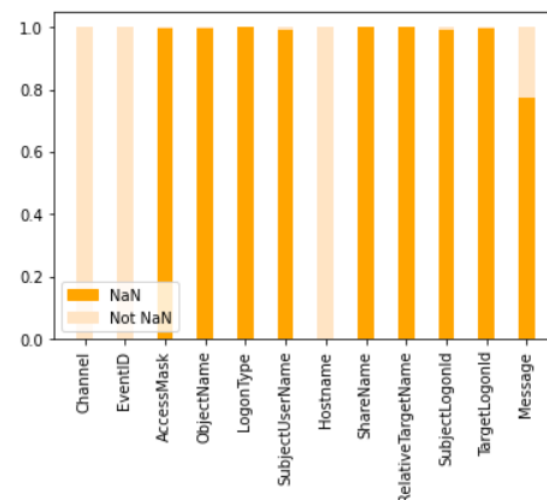
Percentages of missing data in Datasets for "Alternate PowerShell Hosts":

Channel:0/2744
 EventID:0/2744
 Message:1835/2744
 Description:2395/2744
 ImageLoaded:2400/2744
 Image:1332/2744
 PipeName:2728/2744



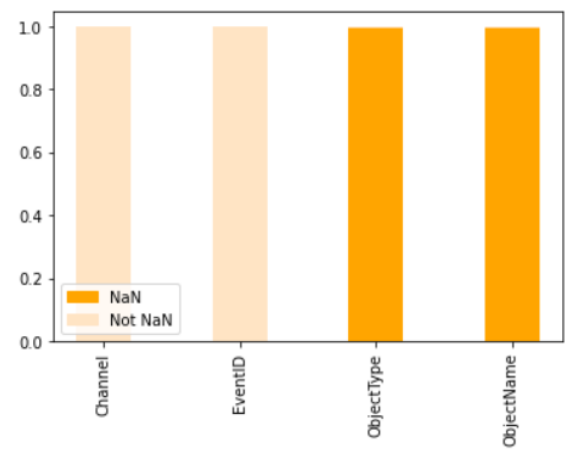
Percentages of missing data in Datasets for "Domain DPAPI Backup Key Extraction":

Channel:0/7887
 EventID:0/7887
 AccessMask:7866/7887
 ObjectName:7870/7887
 LogonType:7881/7887
 SubjectUserName:7825/7887
 Hostname:0/7887
 ShareName:7882/7887
 RelativeTargetName:7883/7887
 SubjectLogonId:7825/7887
 TargetLogonId:7869/7887
 Message:6118/7887



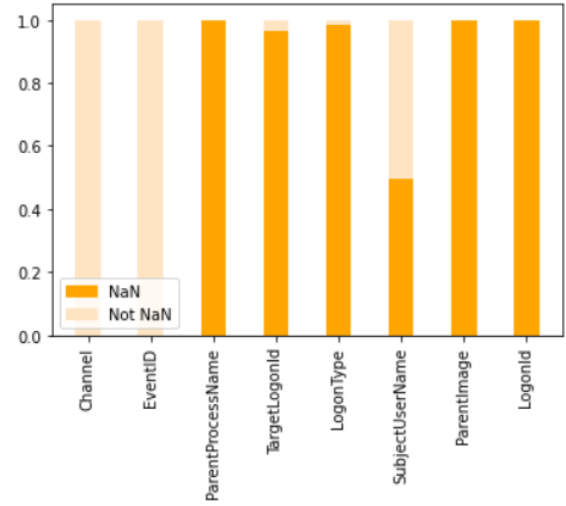
Percentages of missing data in Datasets for "SysKey Registry Keys Access":

Channel:0/12349
EventID:0/12349
ObjectType:12303/12349
ObjectName:12317/12349



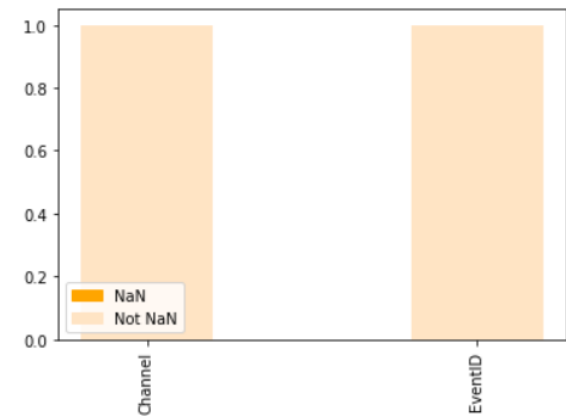
Percentages of missing data in Datasets for "WMI Win32_Process Class and Create Method for Remote Execution":

Channel:0/6383
EventID:0/6383
ParentProcessName:6376/6383
TargetLogonId:6165/6383
LogonType:6292/6383
SubjectUserName:3162/6383
ParentImage:6378/6383
LogonId:6378/6383



Percentages of missing data in Datasets for "WMI Eventing":

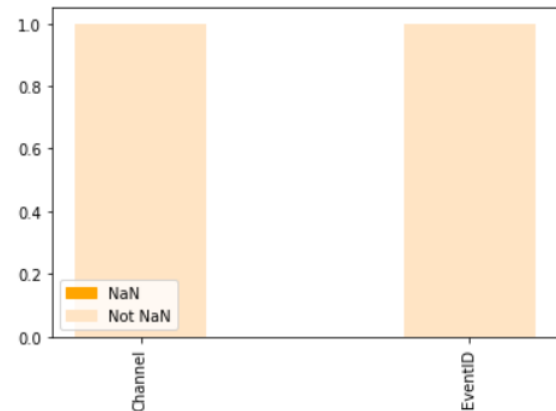
Channel:0/79896
EventID:0/79896



Percentages of missing data in Datasets for "Local Service Installation":

Channel:0/4348

EventID:0/4348



Percentages of missing data in Datasets for "Remote Service creation":

Channel:0/4348

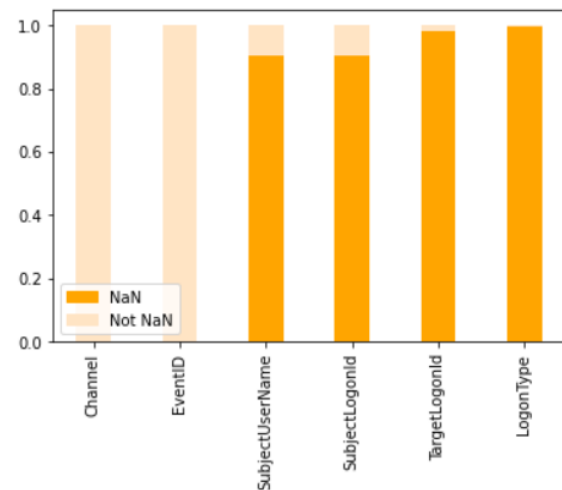
EventID:0/4348

SubjectUserName:3933/4348

SubjectLogonId:3933/4348

TargetLogonId:4279/4348

LogonType:4323/4348



Percentages of missing data in Datasets for "Remote Service Control Manager Handle":

Channel:0/4570

EventID:0/4570

ObjectType:4505/4570

ObjectName:4507/4570

AccessMask:4507/4570

SubjectLogonId:4356/4570

PrivilegeList:4517/4570

Application:4350/4570

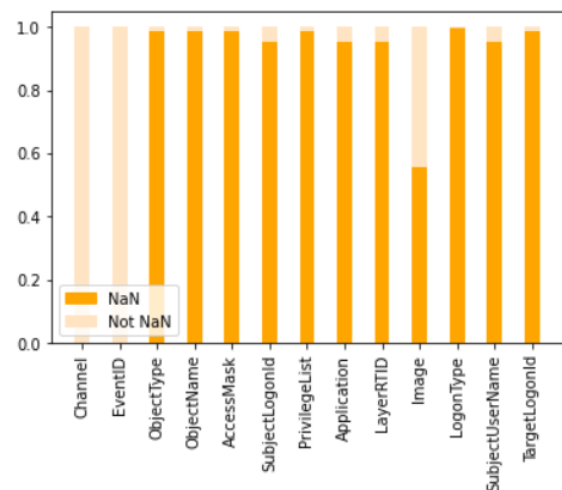
LayerRTID:4350/4570

Image:2543/4570

LogonType:4547/4570

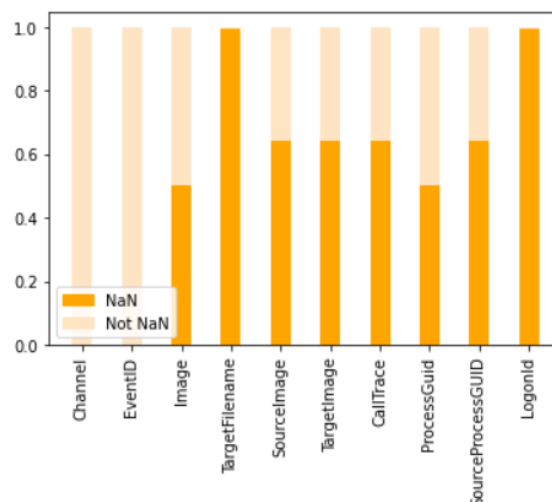
SubjectUserName:4356/4570

TargetLogonId:4517/4570



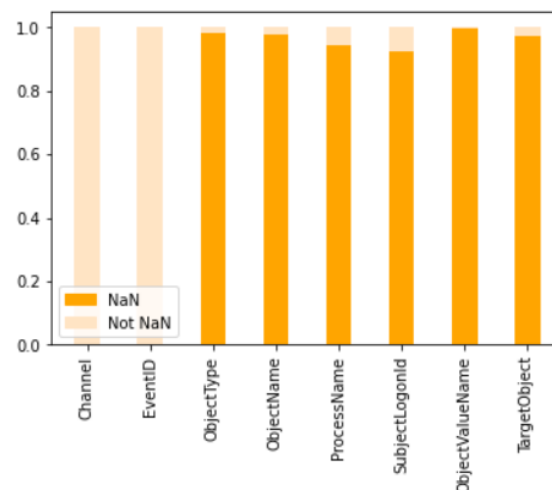
Percentages of missing data in Datasets for "Remote Interactive Task Manager LSASS Dump":

Channel:0/5489
 EventID:0/5489
 Image:2764/5489
 TargetFilename:5450/5489
 SourceImage:3526/5489
 TargetImage:3526/5489
 CallTrace:3527/5489
 ProcessGuid:2764/5489
 SourceProcessGUID:3527/5489
 LogonId:5472/5489



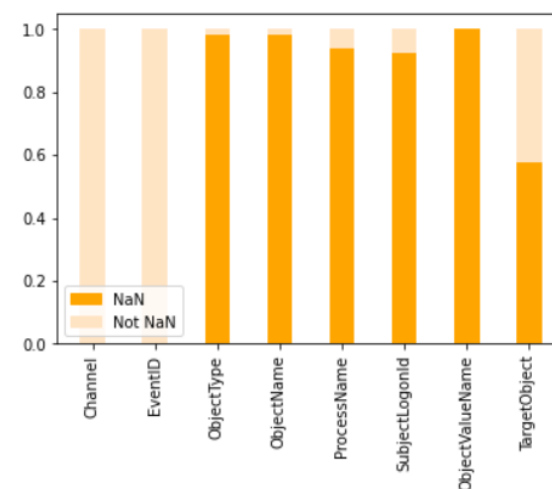
Percentages of missing data in Datasets for "Registry Modification for Extended NetNTLM Downgrade":

Channel:0/1635
 EventID:0/1635
 ObjectType:1604/1635
 ObjectName:1598/1635
 ProcessName:1542/1635
 SubjectLogonId:1508/1635
 ObjectValueName:1629/1635
 TargetObject:1592/1635



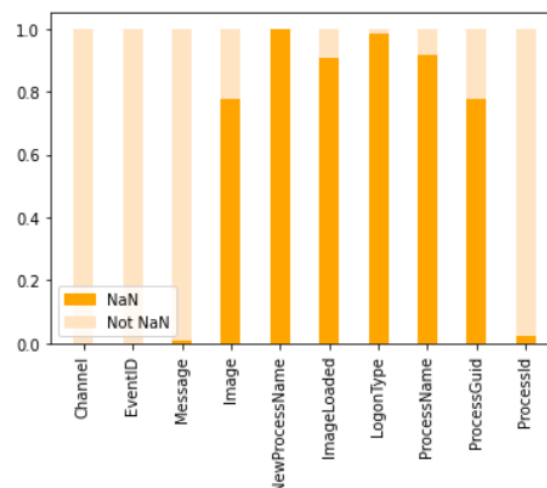
Percentages of missing data in Datasets for "Access to Microphone Device":

Channel:0/6202
 EventID:0/6202
 ObjectType:6093/6202
 ObjectName:6092/6202
 ProcessName:5809/6202
 SubjectLogonId:5731/6202
 ObjectValueName:6200/6202
 TargetObject:3590/6202



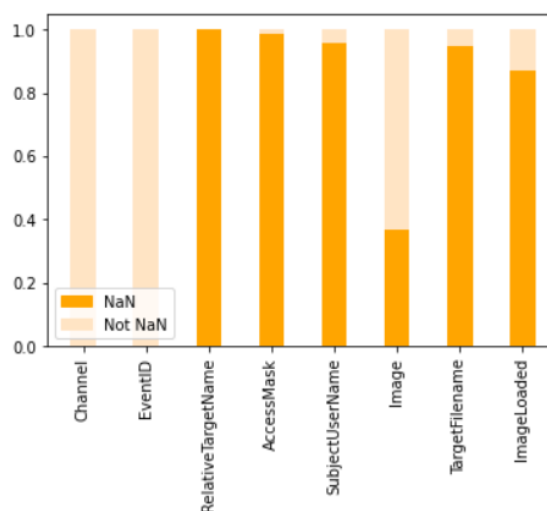
Percentages of missing data in Datasets for "Remote WMI ActiveScriptEventConsumers":

Channel:0/3719
 EventID:0/3719
 Message:25/3719
 Image:2879/3719
 NewProcessName:3714/3719
 ImageLoaded:3376/3719
 LogonType:3662/3719
 ProcessName:3409/3719
 ProcessGuid:2879/3719
 ProcessId:95/3719



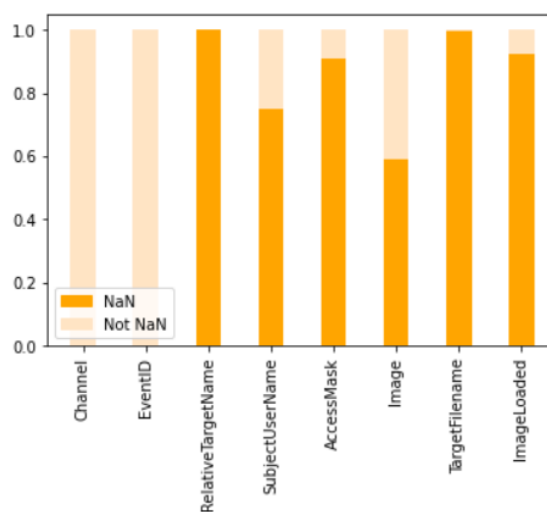
Percentages of missing data in Datasets for "Remote DCOM IErtUtil DLL Hijack":

Channel:0/37814
 EventID:0/37814
 RelativeTargetName:37808/37814
 AccessMask:37239/37814
 SubjectUserName:36261/37814
 Image:13988/37814
 TargetFilename:35799/37814
 ImageLoaded:32974/37814



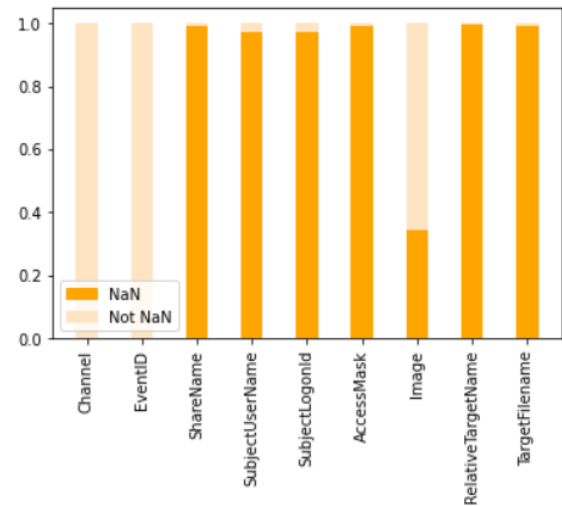
Percentages of missing data in Datasets for "Remote WMI Wbemcomn DLL Hijack":

Channel:0/8947
 EventID:0/8947
 RelativeTargetName:8944/8947
 SubjectUserName:6727/8947
 AccessMask:8150/8947
 Image:5292/8947
 TargetFilename:8920/8947
 ImageLoaded:8272/8947



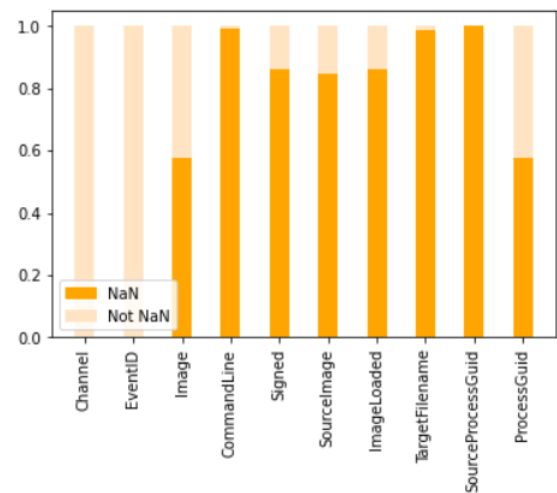
Percentages of missing data in Datasets for "SMB Create Remote File":

Channel:0/506
 EventID:0/506
 ShareName:501/506
 SubjectUserName:492/506
 SubjectLogonId:492/506
 AccessMask:501/506
 Image:173/506
 RelativeTargetName:503/506
 TargetFilename:501/506



Percentages of missing data in Datasets for "Wuauclt CreateRemoteThread Execution":

Channel:0/1326
 EventID:0/1326
 Image:766/1326
 CommandLine:1318/1326
 Signed:1143/1326
 SourceImage:1125/1326
 ImageLoaded:1143/1326
 TargetFilename:1307/1326
 SourceProcessGuid:1325/1326
 ProcessGuid:766/1326



Bibliography

- [1] Ehsan Aghaei and Gursel Serpen. Ensemble classifier for misuse detection using n-gram feature vectors through operating system call traces. *International Journal of Hybrid Intelligent Systems*, 14(3):141–154, 2017.
- [2] James P Anderson. Computer security threat monitoring and surveillance. *Technical Report*, James P. Anderson Company, 1980.
- [3] Jim Brazell and Michael Bettersworth. High performance computing. *Technical brief*, Texas State Technical College–TSTC Forecasting, 2008.
- [4] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. Classification and regression trees. wadsworth int. Group, 37(15):237–251, 1984.
- [5] 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.
- [6] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.
- [7] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, et al. Recent advances in deep learning for speech research at microsoft. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8604–8608. IEEE, 2013.
- [8] Dorothy E Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987.
- [9] Charles Elkan. Results of the kdd’99 classifier learning. *Acm Sigkdd Explorations Newsletter*, 1(2):63–64, 2000.
- [10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
- [12] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1322–1328. IEEE, 2008.

- [13] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.
- [14] Feng Hu and Hang Li. A novel boundary oversampling algorithm based on neighborhood rough set model: Nrsboundary-smote. *Mathematical Problems in Engineering*, 2013:1–10, 2013.
- [15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.
- [16] Jin Kim, Nara Shin, Seung Yeon Jo, and Sang Hyun Kim. Method of intrusion detection using deep neural network. In *2017 IEEE international conference on big data and smart computing (BigComp)*, pages 313–316. IEEE, 2017.
- [17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
- [19] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. In *Recent Advances in Intrusion Detection: 6th International Symposium, RAID 2003, Pittsburgh, PA, USA, September 8-10, 2003. Proceedings 6*, pages 173–191. Springer, 2003.
- [20] Sandeep Kumar and Eugene H Spafford. An application of pattern matching in intrusion detection. 1994.
- [21] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Convolutional neural networks for visual recognition. Available in <http://cs231n.github.io/convolutional-networks>, 2015.
- [22] Steven McElwee, Jeffrey Heaton, James Fraley, and James Cannady. Deep learning for prioritizing and responding to intrusion detection alerts. In *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*, pages 1–5. IEEE, 2017.
- [23] Anthony J Myles, Robert N Feudale, Yang Liu, Nathaniel A Woody, and Steven D Brown. An introduction to decision tree modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 18(6):275–285, 2004.
- [24] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- [25] Robert C Newman. *Computer security: Protecting digital resources*. Jones & Bartlett Publishers, 2009.
- [26] Yang-jia Ou, Ying Lin, Yan Zhang, and Yang-jia Ou. The design and implementation of host-based intrusion detection system. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, pages 595–598, 2010.
- [27] JR Quinlan. C4. 5 programs for machine learning. morgan kaufmann, san mateo, california, 1993.
- [28] Kajal Rai, M Syamala Devi, and Ajay Guleria. Decision tree based algorithm for intrusion detection. *International Journal of Advanced Networking and Applications*, 7(4):2828, 2016.
- [29] John H Ring IV, Colin M Van Oort, Samson Durst, Vanessa White, Joseph P Near, and

- Christian Skalka. Methods for host-based intrusion detection with deep learning. *Digital Threats: Research and Practice (DTRAP)*, 2(4):1–29, 2021.
- [30] Roberto Rodriguez. Threat hunter playbook, 2022. Available at <https://threathunterplaybook.com/intro.html>.
- [31] Roberto Rodriguez and Jose Rodriguez. Security datasets, 2022. Available at <https://securitydatasets.com/introduction.html>.
- [32] Ben D Sawyer, Victor S Finomore, Gregory J Funke, Vincent F Mancuso, Matthew E Funke, Gerald Matthews, and Joel S Warm. Cyber vigilance: effects of signal probability and event rate. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 58, pages 1771–1775. Sage Publications Sage CA: Los Angeles, CA, 2014.
- [33] Jaydip Sen and Sidra Mehtab. Machine learning applications in misuse and anomaly detection. *Security and privacy from a legal, ethical, and technical perspective*, page 155, 2020.
- [34] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. In *Technical report*. The MITRE Corporation, 2018.
- [35] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.
- [36] Benjamin X Wang and Nathalie Japkowicz. Imbalanced data set learning with synthetic samples. In *Proc. IRIS machine learning workshop*, volume 19, page 435, 2004.
- [37] Kevin S Woods, Christopher C Doss, Kevin W Bowyer, Jeffrey L Solka, Carey E Priebe, and W Philip Kegelmeyer Jr. Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(06):1417–1436, 1993.
- [38] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. *Advances in Neural Information Processing Systems*, 32, 2019.