Early Detection of Data Exfiltration Attacks using Gradient Boosted Decision Trees

by Anthony Agustin Balagao

Master of Science in Information Systems Technology, May 2013,
The George Washington University School of Business
Technical Degree in Computer Technology, March 1990,
Technological University of the Philippines

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

January 10, 2025

Praxis directed by

John M. Fossaceca
Professorial Lecturer of Engineering and Applied Science

Shahryar Sarkani
Adjunct Professor of Engineering and Applied Science

The School of Engineering and Applied Science of The George Washington University certifies that Anthony Agustin Balagao has passed the Final Examination for the degree of Doctor of Engineering in Cybersecurity Analytics as of January 10, 2025. This is the final and approved form of the Praxis.

Early Detection of Data Exfiltration Attacks using Gradient Boosted Decision Trees

Anthony Agustin Balagao

Praxis Research Committee:

John M. Fossaceca, Professorial Lecturer of Engineering and Applied Science
Praxis Director

J.P. Blackford, Associate Professor of Engineering and Applied Science
Committee Member

Amir Etemadi, Associate Professor of Engineering and Applied Science Praxis
Chair

© Copyright 2024 by Anthony Agustin Balagao
All rights reserved.

Dedication

The author wishes to dedicate this research to his wife and children for their unconditional love and unwavering support. For Barkley.

Acknowledgements

The author wishes to thank the esteemed praxis director and co-director Professor Fossaceca and Professor Sarkani, Committee Member, Committee Chair, Faculty and Staff of The George Washington University School of Engineering and Applied Sciences for their invaluable guidance and making this achievement a reality, colleagues at the DHS CISA CMIP, CISA Threat Hunt and Forensics, the Raytheon Product Security Team Director Sung Oh and Raytheon CTO Juan De Bedeau for making this journey possible. Special mention, Nightwing CTO John Xereas, Northrop Grumman CTO Staff Dr. Jeff Yu. Thank you for all your support.

Abstract of Praxis

Early detection of Data Exfiltration attacks using Gradient Boosted Decision Trees

The technology sector has been a frequent target of data exfiltration attacks in recent years. While several incidents go unreported, several high-profile cases have become known. Various threat actors have targeted U.S. tech companies, including nation-state hackers, cybercriminals, and hacktivists. (Sabir, et al. 2020). “Gradient Boosted Decision Trees (GBDT) also known as Gradient Boosting Machines (GBMs)“ (Friedman, 2001) can be effectively employed to detect data exfiltration attempts by sequentially adding weak predictive models, it can accurately identify anomalous network traffic patterns that may indicate malicious activity. “GBDTs can provide insights into the importance of unique features in predicting data breaches.” (Friedman, 2001). “Neural networks can be trained to identify unusual patterns in network traffic or user behavior, which may indicate potential data exfiltration attempts.” (Dolhopolov, 2024)

Table of Contents

Dedication iv

Acknowledgements v

Abstract of Praxis vi

List of Figures x

List of Tables xi

List of Acronyms xii

Chapter 1—Introduction 1

1.1 Background 1

1.2 Research Motivation 2

1.3 Problem Statement 2

1.4 Thesis Statement 3

1.5 Research Objectives 3

1.6 Research Questions and Hypotheses 3

1.7 Scope of Research 4

1.8 Research Limitations 5

1.9 Organization of Praxis 5

Chapter 2—Literature Review 7

2.1 Introduction 7

2.2 DNS Exfiltration 8

2.3 Bot-IoT Dataset 8

2.4 Gradient Boosting Machine 10
2.5 Backpropagation in Neural Networks 11
2.6 Data Exfiltration Detection using Autoencoders. 12
2.7 Detecting DNS Exfiltration using Unsupervised Learning 13
2.8 Data Theft in Unmanned Aerial Vehicles (UAVs) 14
2.9 XGBoosted Neural Networks 15
2.10 Neural Network Threat Detection Systems for Data Breach Protection 16
2.11 Network Classification using CNN vs RNN. 18
2.12 Ada Boosted Neural Networks (ABNN) 19
2.13 Imbalanced Classification for Botnet Detection in Internet of Things 20
2.14 IoT-botnet Detection using Fuzzy Rule Interpolation 20
2.15 Simulating attacks using MITRE Framework 22
2.16 Conclusion 23
Chapter 3—Methodology 25
3.1 Introduction 25
3.2 Attack Dataset Generation 25
3.3 Dataset Composition 30
3.4 Dataset Features 31
3.5 Methodology Diagram 33
3.5.1 Methodology Step 1 – Data Files Selection 39
3.5.2 Methodology Step 2 – Data Preprocessing 42
3.5.3 Methodology Step 3 – Build Neural Network models 46
3.5.4 Methodology Step 4 – Build GBDT models 49

3.5.5 Methodology Step 5 – Apply SMOTE and Tune Hyperparameters 50
3.5.6 Methodology Step 6 – Compare prediction results for each model 51
3.6 Conclusion 54
Chapter 4—Results 56
4.1 Introduction 56
4.2 Convolutional Neural Networks (CNN) Results 56
4.3 Deep Neural Networks (DNN) Results 57
4.4 Multilayer Perceptron (MLP) Results 58
4.5 Gradient Boosted Decision Tree Results 59
4.6 XGBoost Results 62
4.7 Overall Classification Results 63
4.8 Conclusion 65
Chapter 5—Discussion and Conclusions 66
5.1 Discussion 66
5.2 Conclusions 67
5.3 Contributions to the Body of Knowledge 68
5.4 Future Research 70
5.5 Future Applications of this Research 73
References 77
Appendix A 84
1.1 Multilayer Perceptron Neural Network 84
1.2 Deep Neural Network 84
1.3 Convolutional Neural Network 85

List of Figures

Figure 1 – Dataset Composition 30

Figure 2 – Methodology Diagram..... 34

Figure 3 – Data Files Selection..... 39

Figure 4 – Dataset Preprocessing..... 43

Figure 5 – Build Neural Network Models 46

Figure 6 – Build GBDT models..... 49

Figure 7 – Apply SMOTE and Tune Hyperparameters 50

Figure 8 – Compare prediction results from each model..... 51

Figure 9 – CNN confusion matrix. 56

Figure 10 – DNN confusion matrix. 57

Figure 11 – MLP confusion matrix..... 59

Figure 12 – GBDT confusion matrix. 60

Figure 13 – XGBoost confusion matrix..... 62

List of Tables

Table 1 – Dataset Features 31

Table 2 – Data Exfiltration Detection Performance Comparison 64

List of Acronyms

CNN	Convolutional Neural Network
DNN	Deep Neural Network
GBNN	Gradient Boosted Neural Network
ABNN	Ada Boosted Neural Network
XGBNN	Xtreme Gradient Boosted Neural Network
MLP	Multilayer Perceptron
TPR	True Positive Rate
TNR	True Negative Rate
FPR	False Positive Rate
FNR	False Positive Rate
UNSW	University of New South Wales
IoT	Internet of Things
IaC	Infrastructure as Code
DoS	Denial of Service
ML	Machine Learning
UL	Unsupervised Learning
SL	Supervised Learning
RL	Reinforcement Learning
FRI	Fuzzy Rule Interpolation
MDP	Markov Decision Process
TTP	Tactics, Techniques, Procedures

APT	Advanced Persistent Threats
DNS	Domain Name System
UAV	Unmanned Aerial Vehicle
DMLPNN	Deep Multilayer Perceptron Neural Network
SMOTE	Synthetic Minority Over-sampling Technique
ADASYN	Adaptive Synthetic Sampling
SVM	Support Vector Machine
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
CTI	Continuous Threat Intelligence
ISMS	Information Security Management System
SIEM	Security Information and Event Management
SOAR	Security Orchestration, Automation, and Response
RF	Random Forest
DT	Decision Tree
GBDT	Gradient Boosted Decision Tree
UNSW	University of New South Wales
USA	United States of America

Chapter 1—Introduction

1.1 Background

Data Exfiltration can have a severe impact on national security. Nation state actors often target sensitive data belonging to other nations for a number of purposes. This could include military plans, economic secrets, diplomatic communications, or personal information about key individuals. Exfiltrating data from critical infrastructure sectors like energy, transportation, or finance can cripple essential services and cause widespread chaos. Stealing intellectual property, trade secrets, or financial data can damage a nation's economic competitiveness and stability. Adversaries could use leaked information to sway public opinion, undermine political institutions, or incite social unrest. Protecting against data exfiltration is crucial for national security. This involves strengthening cybersecurity defenses, promoting international cooperation in combating cybercrime, and raising awareness about the risks of data breaches. (Sabir, et al. 2020)

Neural networks consist of individual neurons that are relatively simple, performing basic calculations. However, when interconnected in layers, they can learn complex patterns and make sophisticated decisions. The key to learning lies in adjusting the weights associated with the connections between neurons. These weights determine the strength of the connection and influence the output of the neuron. “Training neural networks involves calculating the error between the predicted output and the actual output, and then adjusting the weights in a way that minimizes this error. Neural networks can learn to recognize patterns by iteratively adjusting the weights through backpropagation to make predictions.” (Hinton, et al. 1986)

Neural networks can be used for detecting and preventing data leaks. Deep multilayer perceptron neural network can be used for threat detection. Neural networks have the ability to detect threat patterns and identify potential data breaches with high accuracy. (Dolhopolov, 2024)

"Gradient Boosted Decision Trees (GBDTs) also known as Gradient Boosting Machines (GBMs) are well-equipped for identifying complex patterns on sequential data." (Friedman, 2001)

1.2 Research Motivation

Data exfiltration attacks can be more subtle and less frequent, making it harder to train detection models with limited examples in the dataset. Even though this dataset aims for realism, it is still a simulated environment. Real-world data exfiltration often involves sophisticated techniques that detection models might not be fully captured in the simulated dataset. Attackers constantly change their methods. Models trained on a static dataset might struggle to detect novel data exfiltration techniques that emerge over time. (Koroniotis, et al. 2020). There are extremely limited number of records available for Data Exfiltration. "This small number of record samples makes it difficult for a machine learning model to learn patterns that can be accurately applied to validation and testing datasets." (Alsarhan, 2023).

1.3 Problem Statement

Data Exfiltration attacks by foreign state actors increased by 12% in 2018 to 23% in 2019, costing an average of 3.86 million USD per incident. (Sabir, et al. 2020)

The rise in nation-state-backed attacks from 12% to 23% in a single year underscores the growing threat posed by these actors. The average cost of 3.86 million

USD per incident demonstrates the substantial financial burden these attacks can impose on organizations.

1.4 Thesis Statement

A detection model is required to detect Data Exfiltration attacks in order to expedite security countermeasures.

A robust detection model is essential for timely identification and response to security threats, including data exfiltration attacks. By automating the detection process, organizations can significantly reduce response times and mitigate potential damage.

1.5 Research Objectives

Our main objective is to create a detection model for the early detection of Data Exfiltration attacks by developing a more efficient methodology for classifying network traffic so that enterprises can integrate the detection model into existing Security Incident Event Management (SIEM) and Security Orchestration Automation Response (SOAR) for security teams to expedite security countermeasures. Enterprises can also integrate with monitoring tools such as Wireshark using LUA scripts.

1.6 Research Questions and Hypotheses

We state the following Research Question and Hypothesis Statements:

RQ1: Which features are best for detecting Data Exfiltration attacks?

RQ2: Can a detection model be constructed to detect Data Exfiltration attacks with at least 95% accuracy?

RQ3: Which algorithm offers the best results for Data Exfiltration attack detection?

H1: Packet header features are the best for detecting Data Exfiltration attacks compared to using packet payload.

H2: A model can be constructed to detect Data Exfiltration attacks with at least 95% accuracy.

H3: Gradient Boosted Decision Tree algorithm is the best algorithm for detecting Data Exfiltration compared to Neural Networks.

1.7 Scope of Research

"Data exfiltration is a type of attack that involves an adversary targeting a remote machine to gain access to data without authorization." (Sabir, 2021). The attacker downloads this data to their own machine. Weaknesses in Windows 7 and Ubuntu Server were used as target machines in which vulnerabilities were exploited. In order to evaluate the detection model, we also cover other types of attacks. OS Scan is a type of network scanning that involves identifying open ports on a system's operating system. Similar to OS scan, Service Scan refers to a method of identifying open services or ports on a network or system that could be exploited by an attacker. HTTP DOS and TCP DOS are types of Denial of Service (DoS) attacks where the attacker floods the targeted machine with HTTP or TCP traffic to overwhelm its resources, rendering it unavailable. The UNSW network testbed is a realistic, simulated environment for testing network security solutions and researching new attack techniques. This allows control and experiment with various network configurations and attack scenarios without risking real-world systems. The network can be configured with different network topologies, devices (routers, switches), and applications to simulate diverse network environments. The network can

manage millions of simulated users and traffic flows, making it suitable for testing large-scale attacks. (Koroniotis, et al. 2018)

1.8 Research Limitations

"There is a lack of standardized methods or approaches for detecting data exfiltration using machine learning suggesting the need for further research and development in this area. The datasets used for evaluating data exfiltration countermeasures do not provide sufficient information on the quality or completeness of these datasets. This can limit the effectiveness of machine learning models trained on these datasets. Furthermore, the performance and accuracy of Machine Learning models is dependent on the available features, quality of datasets, machine learning algorithm, and evaluation method." (Sabir, et al. 2020). This implies that the current models have inherent flaws or limitations in these areas. A review of external attack vectors suggests that there is still much to be learned about how data exfiltration attacks occur and how they can be effectively defended against.

1.9 Organization of Praxis

Chapter One explains the background of data exfiltration attacks, the motivation for this research, problem statement that need to be addressed, thesis statement stating the action required, "research objectives, the research questions, research hypotheses, scope of research, and research limitations." (Koroniotis, 2019) Chapter Two is a literature review of research performed on data breach attacks as well as detection methods and techniques. Chapter Three explains the methodology of developing the detection model. Chapter Four explains the results of the experimentation. Chapter Five contains

discussion and the conclusion as well as recommendations for future research for detecting industry specific exfiltration attacks.

Chapter 2—Literature Review

2.1 Introduction

Detecting data exfiltration using machine learning requires labeled datasets. Both supervised learning (which requires both good and bad examples) and one-class classification (which needs a dataset of only good examples) struggle because real-world datasets with known exploits are scarce. Researchers often resort to synthetic or curated datasets, but these can introduce bias, making the model less effective in real-world scenarios. Additionally, the imbalance between legitimate and malicious traffic (often many more benign queries than malicious ones) further complicates training accurate models. In the supervised machine learning approach for detecting data exfiltration, models require both benign and malicious data samples. However, obtaining realistic network datasets with known exploits is often difficult because they are rarely open to the public. To solve this problem, studies have used synthetic datasets, however this method can introduce bias into the model. “Also, research suggests that machine learning algorithms may not perform as well when dealing with an imbalanced dataset, which is most common for data exfiltration datasets. A one-class classifier can only have benign data which is different from the supervised approach which requires both benign and malicious data samples.” (Joback, et al. 2020). In this method, the model is trained on normal or benign traffic with the assumption that most network traffic is benign. The model is trained on the patterns and characteristics of benign traffic which it uses to

classify new network traffic as either benign or potentially malicious based on how closely they resemble the learned patterns.

2.2 DNS Exfiltration

Data exfiltration with "low and slow" attacks that have a low throughput can still cause considerable damage. Low throughput exfiltration malware can be difficult to detect due to its slow data exchange rate. Periods of inactivity followed by brief periods of activity make it challenging to establish detection coverage against these types of attacks. “The need for a solution that can manage streaming DNS traffic and detect and deny requests to domains used for data exfiltration.” (Nadler, et al. 2018)

Gaps: The lack of a standard dataset for DNS tunneling makes it difficult to compare and evaluate different approaches. “Minimizing high false positive rates can lead to unnecessary alarms and alert fatigue.” (Abualghanam, et al. 2023). Dataset challenges for detecting data exfiltration include scarcity of publicly available datasets which make it difficult to compare and evaluate different approaches. Existing datasets are outdated (approximately one-decade old), which makes them unable to represent novel and sophisticated attack vectors. Datasets created privately for research purposes are often not made public due to concerns about privacy, intellectual property, competitive advantage, and the resources spent on creating them which also presents the problem of integrity. These dataset challenges highlight the difficulties in developing effective detection methods and systems for data exfiltration, emphasizing the need for high-quality evaluation datasets that can accurately represent real-world scenarios.

2.3 Bot-IoT Dataset

Attackers often use botnets to steal data, so analyzing their communication patterns and activities could reveal signs of data being moved out of a system. Understanding botnet behavior (which the Bot-IoT dataset helps with) is a crucial step in detecting data exfiltration. Analyzing network traffic for unusual patterns, such as substantial amounts of data transferred to unknown destinations or connections made to known malicious servers, could indicate data exfiltration. Examining communication between infected devices and the botnet's command and control server can reveal instructions related to data extraction or transfer. Monitoring large file transfers from internal systems to external locations, especially if they involve sensitive data types, could be a red flag. Observing changes in system behavior, such as increased CPU usage or disk activity, could suggest that data is being processed and moved out of the system. (Koroniotis et al., 2019).

Gaps: One of the Bot-IoT dataset's drawbacks is that it has an imbalance nature of data. This means that there is “a significant difference in the number of samples between the majority class and the minority class. In this case, around 87.35% of the data in the UNSW-NB15 dataset is non-anomalous (normal), whereas only 0.013% of the data in the Bot-IoT dataset is non-anomalous (normal).” (Zeeshan, et al. 2022). This imbalance can lead to biased predictions when training a machine learning model. For instance, if a classifier is trained solely on the UNSW-NB15 dataset, it will likely be exceptionally good at predicting normal packets because there are more examples of normal packets than any other type. However, it will not be as effective at predicting abnormal packets (such as DDoS or DoS) because there are so few examples of these types. Similarly, if a classifier is trained solely on the Bot-IoT dataset, it will likely be exceptionally good at

predicting DDoS and DoS packets because there are more examples of these types than any other type. However, it will not be as effective at predicting normal packets because there are so few examples of these.

2.4 Gradient Boosting Machine

"A Gradient Boosting Machine is an ensemble learning method that combines weak learners specifically decision trees of limited depth, which perform better compared to random guessing. The Loss Function measures the difference between the predictions and actual values of the prediction. These include mean squared error (for regression) and cross-entropy (for classification). The Stochastic Gradient Descent (SGD) is an optimization algorithm to minimize the loss function." (Friedmann, 2001). With each iteration, SGD determines the gradient of the loss with respect to the predictions and then updates the parameters to reduce the loss. Gradient Boosting Machine builds an additive model by sequentially adding weak learners. Each new learner is trained to predict the residual of the loss function which controls the contribution of each learner to the final model. The more iterations, the smaller learning rate is required but this can lead to a more or less accurate model. Gradient Boosting Machine can be used for network classifications by applying it to the features extracted from the network data. The process involves training a series of weak decision trees or linear models on the residuals between the true and actual values, incrementally building up a stronger model. "This process helps to learn complex patterns and make accurate predictions." (Friedman, 2001). There are several implementations of Gradient Boosting Machine that can be used for network classifications, such as Gradient Boost, LightGBM, and CatBoost. These libraries provide various optimizations that help improve training time and performance on large datasets.

Gaps: A variation of the Gradient Boosting Machine (Stochastic Gradient Boosting) introduces randomness into the training process by using subsamples. Subsamples help reduce the model from overfitting. Each iteration is faster than in a standard Gradient Boosting Machine for smaller subsets of data. Instead of using the entire training dataset to update the model at each iteration, Stochastic Gradient Boosting uses random subsets called subsamples of the data which are typically much smaller than the full dataset. “Subsampling helps to prevent overfitting, which is a common problem where the model becomes too specialized in the training data and performs poorly on new, unseen data.” (Friedman, 2001) Benefits of Stochastic Gradient Boosting include reduced overfitting. Stochastic Gradient Boosting can be more resistant to noisy data because it averages out the effects of individual data points.

2.5 Backpropagation in Neural Networks

Deep neural networks can be used to classify network traffic using individual neurons that are relatively simple, performing basic calculations. However, when interconnected in layers, they can learn complex patterns and make sophisticated decisions. “The key lies in adjusting the weights associated with the connections between neurons which in turn determine the how strong the connection and influence the output of the neuron. Training neural networks involves calculating the error between the true output and the actual output and adjusting the weights to minimize the error. By incrementally adjusting the weights, the neural network can recognize patterns to make accurate predictions and solve highly complex problems.” (Dolhopolov, 2024). Neural networks such as Deep Neural Networks can be used for detecting and preventing data

leaks, detect complex threat patterns and identify potential data breaches with high accuracy.

Gaps: Boltzmann machines have layers of interconnected nodes called neurons which can be either visible or hidden. Visible nodes represent the input data, while hidden nodes learn representations and patterns within the data. Unlike feedforward neural networks, “Boltzmann machines maximize the probability of learning from the training data by adjusting the weights between nodes. This process involves sampling from the network's probability distribution then comparing it to the actual training samples. Boltzmann machines introduce randomness into their activations, allowing them to explore different patterns and configurations in the data.” (Hinton, et al. 1986). The connections between nodes in a Boltzmann machine are undirected (bidirectional), meaning information can flow in both directions. This contrasts with traditional neural networks, which typically have direct connections. Boltzmann machines excel at learning hierarchical data, making them useful for feature extraction and dimensionality reduction. Traditional neural networks like those using backpropagation use feedforward, meaning “information flows only in one direction from input to output.” (Hinton, et al. 1986). In Boltzmann Machines, on the other hand, they are stochastic and operate more like a stable system in thermal equilibrium. They sample from probability distributions and do not have a clear "input" to "output" pathway. Backpropagation is based on “calculating the gradients of the error function with respect to each weight in the network. “ (Hinton, et al. 1986). This allows to adjust weights incrementally using gradient descent to minimize the error.

2.6 Data Exfiltration Detection using Autoencoders.

“Autoencoder consists of an encoder that converts the input data into a lower-dimensional representation and a decoder that reconstructs the original input from the internal representation which can be used to detect anomalies in network metadata. The autoencoder is trained only by using benign training dataset then be fed with a test dataset to detect anomalies which can also be applied to detecting data exfiltration in network traffic.” (Willems, et al. 2023)

Gaps: “Autoencoders and Variational Autoencoders (VAEs) are types of unsupervised deep learning which can learn from data in a way that allows them to reproduce the original data.” (Al-shehari, 2024). They can be trained on "normal" user behavior. Anomalies in the dataset serve as potential insider threats resulting in a more balanced and accurate model. Imbalanced addressing can be used to mitigate class imbalance issues in the detection model. “The Synthetic Minority Over-sampling Technique (SMOTE) is a technique to address class imbalance by generating synthetic samples of the minority class thereby helping to balance the dataset and improve model performance.” (Chawla, 2002). The goal of imbalanced addressing is to create a balanced dataset which allows the model to learn from both classes without being biased towards the majority class. This leads to improved performance, reduced false positives, and enhanced accuracy. Combining a convolutional neural network (CNN) with an imbalance addressing technique can also be “employed to analyze spatial data or a grid-like structure such as time series data.” (Al-shehari, 2024).

2.7 Detecting DNS Exfiltration using Unsupervised Learning

Unsupervised Learning (UL) such as clustering or anomaly detection algorithms can also be used to identify data exfiltration in network traffic. Unsupervised Learning

(UL) can group similar network traffic patterns together to identify anomalies. Unsupervised learning is often more effective than supervised learning for detecting data exfiltration because it can manage large amounts of “unstructured data and identify patterns that might not be apparent through other methods. However, the goal is to learn from experience and adapt to new threats.” (Joback, 2020). Unsupervised learning is often more effective when the dataset is high-dimensional or unstructured in which the “goal is to identify patterns or anomalies in the data.” (Joback, 2020). The imbalance between legitimate and malicious traffic is significant.

Gaps: Supervised learning is more suitable for DNS exfiltration detection. Fourteen out of 92 studies used supervised learning methods to extract features which implies that there is a labeled dataset available for training machine learning models. This suggests that supervised learning can be used for DNS exfiltration detection where “the model is trained on labeled examples of normal and malicious traffic. Unsupervised learning typically requires a large amount of unlabeled data to identify traffic patterns.” (Sabir, et al. 2020). However, using labeled data for training models warranted learning the preferred approach being used for detecting DNS exfiltration.

2.8 Data Theft in Unmanned Aerial Vehicles (UAVs)

Attackers could gain access to sensitive data stored on the UAV or transmitted between it and other systems (e.g., GPS coordinates, mission plans, video footage). Malware could be injected into the UAV's system, allowing attackers to remotely control its actions or steal data. Overwhelming the UAV with traffic could disrupt its operation and prevent it from completing its mission. An attacker could impersonate a legitimate user or command center to manipulate the UAV's behavior or gain unauthorized access.

Intercepting communications between the UAV and other devices to steal information or track its movements. UAV keylogger attacks are also a serious threat because they allow attackers to steal sensitive data which could include flight plans, military strategies, or personal information of people operating the drone. An attacker could use a keylogger to gain access to the controls of a drone and potentially redirect it, causing damage or disrupting operations. A compromised drone could be used to spy on targets, deliver contraband, or even launch attacks. They demonstrate how various actors can use these attacks, from criminals to nation-states, to achieve their goals. (Molina, 2023)

Gaps: Challenges in UAV Intrusion Detection System (IDS) include achieving a good balance between detection latency which affects agility as well as computational cost and communication overhead. Balancing computational cost with accuracy and security requirements. Mitigating power consumption and performance degradation caused by high implementation overhead. Accurately observing and modeling threat and attack behaviors to enhance defense strategies. Developing effective threat assessment policies to mitigate vulnerabilities and risks. Balancing conflicting goals of maximum network throughput and minimum cost. Developing lightweight IDS mechanisms for battery-operated, resource-constrained UAVs. Quickly detecting attacks and responding effectively to minimize damages or vulnerabilities. (Choudhary, et al. 2018)

2.9 XGBoosted Neural Networks

XGB-NN is a novel architecture that combines Gradient Boosted Machines with "feedforward neural networks designed to handle tabular data" (Sarkar, 2022) and provides good interpretability compared to traditional neural networks. XGB-NN works by using "an optimization technique called Boosted Gradient Descent, which initializes

the weights of each layer with the feature importance of a Gradient Boosted Machine." (Sarkar, 2022). The neural network is trained in two steps. First, it updates the weights of each layer then it applies Gradient Boost to the first few layers. XGB-NN has been "evaluated on various datasets, and the results show that it can achieve high training and testing accuracy." (Sarkar, 2022). Extreme Boosted Neural Network could potentially be designed to learn patterns from labeled datasets of normal network traffic or system activity. By feeding new and unseen data into XGB-NN, the model flags an event as anomalous, further investigation and alerting mechanisms can be triggered. (Sarkar, et al. 2022)

Gaps: XGBoosted Convolutional Neural Networks (XGB-CNN) combines the strengths of convolution neural networks and XGBoost. XGB-CNN improves the performance of CNNs on tasks such as image classification. In XGB-CNN, "CNN is used as a feature extractor to generate features that are then fed into XGBoost" (Tongsuwan, et al. 2021) to perform the final classification task. The use of CNN as a feature extractor allows for the benefits of both models. The robustness and scalability of XGBoost, combined with the ability of CNNs to learn complex spatial patterns in images. XGB-CNN advantages include improved accuracy, robustness to overfitting, and better handling of large datasets compared to traditional CNN architecture.

2.10 Neural Network Threat Detection Systems for Data Breach Protection

An advanced "neural network-based system can be used" (Dolhopolov, 2024) for detecting and preventing data leaks. Deep neural network can be used for threat detection. Neural networks can enable the system to detect complex threat patterns and identify potential data breaches with high accuracy. This multilayer perceptron can be an

effective deep learning threat detection system for data breaches, focusing primarily on the deep multilayer perceptron neural network. “Gradient Boosting Machine can improve the performance of Deep Neural Network (DNN)” (Dolhopolov, 2024) for detecting potential data breaches. The use of Gradient Boosting Machine helps to identify “complex patterns in the data that may be difficult for the DNN model to capture, leading to improved accuracy and completeness of the results.” (Dolhopolov, et al. 2024)

Gaps: One potential drawback is that these models can be expensive to train computationally which may limit their applicability in some cases. Additionally, DNNs can be “sensitive to the choice of hyperparameters, such as the number of weak learners” (Badirli, 2020) used for training and the regularization parameters applied. Overall, while ensemble methods like Gradient Boosting Machine offer many benefits, it is important to carefully consider their limitations and choose the appropriate method based on the specific problem being addressed. Gradient Boosted Neural Networks (GBNN) can be used to detect data exfiltration by leveraging Neural Networks’ ability to learn complex patterns in data and improve over time through iterative training. Gradient Boosted Neural Networks (GBNNs) combine the strengths of Gradient Boosting Machine and neural networks to create a powerful and flexible machine learning model. This hybrid approach can be particularly effective for complex tasks involving sequential or spatial data. GBNNs can provide analysis of the importance of distinctive features in the input data. GBNNs are well-equipped for tasks involving complex patterns and sequential data. (Badirli, et al. 2020)

Gaps: Overfitting can occur when “a model is too complex for the training data, causing it to focus on the noise and random fluctuations rather than the underlying

patterns especially when the number of iterations is high. To mitigate overfitting, techniques like early stopping," (Emami, 2023) regularization, or ensemble methods can be employed. "Training a large-scale GBNN model can be computationally expensive and may require significant computational resources." (Emami, 2023). This is because each iteration involves computing gradients and updating the model parameters, which can be time-consuming. To reduce the computational cost, parallel processing, distributed training, or using specialized hardware (like GPUs) can be employed. As with any complex deep learning model, interpreting the results and understanding how the predictions are made can be challenging. In GBNNs, this might involve understanding how each iteration contributes to the final prediction, which can be difficult without a clear visual representation or explicit explanations from the model. (Emami, et al. 2020).

2.11 Network Classification using CNN vs RNN.

RNNs can manage sequential data with varying lengths, making them suitable for tasks that involve variable-length input sequences. "They can capture long-term dependencies in the data, which makes them useful for tasks such as language modeling. RNNs have been shown to be effective in a variety of applications, including speech recognition," (Mahuri, et al. 2020) machine translation, and text generation.

Gaps: RNNs require a large amount of memory, making them difficult to train on large datasets. They can suffer from the "vanishing gradient" where gradients become smaller as they propagate through the network, making it difficult to train deep RNNs. CNNs are highly effective for handling large amounts of data and takes a relatively short

amount of time to train. They are highly parallelizable, making them efficient to train on multiple CPU cores or GPUs. (Xin, et al. 2020)

2.12 Ada Boosted Neural Networks (ABNN)

AdaBoost with neural networks can improve detection of data exfiltration by leveraging the "strengths of both Gradient Boosting and neural networks." (Schwenk, 2000). A boosted multi-layer network achieved significant improvements in error rates on certain datasets, implying that this approach may be effective for detecting data exfiltration. In general, AdaBoost is a popular algorithm for "improving the performance of machine learning models" (Schwenk, 2000) by iteratively adding weak classifiers to strengthen the overall model. When applied to neural networks, AdaBoost can potentially help improve "the accuracy and robustness of these models" (Schwenk, 2000) in detecting anomalies, such as data exfiltration. possible ways that AdaBoost with neural networks could improve detection of data exfiltration include enhancing anomaly detection capabilities by leveraging the strengths of both Boosted and neural networks, AdaBoost may be able to improve the ability of the model to detect unusual or malicious patterns in data. The Boosted component of the algorithm can help "reduce the impact of noisy or varied data on the overall model," (Schwenk, 2000) making it more effective at detecting data exfiltration. Increasing accuracy with multiple classifiers by combining multiple weak neural network classifiers, AdaBoost can potentially improve the overall accuracy of the detection model. (Schwenk, et al. 2000)

Gaps: although Adaboost on a neural network can learn more complex patterns compared to decision trees, a neural network base estimator can be more expensive computationally compared to traditional Gradient Boosted methods since it requires

training multiple instances of the same neural network. However, this added complexity is often worth it, since Adaboost can lead to significant improvements in performance in certain tasks. In comparison, Adaboost-CNN uses Adaboost to select the weights of each layer. In this case, Adaboost-CNN is a more traditional neural network architecture, and its complexity and performance are comparable to other CNN architectures. (McGinnity, et al.)

2.13 Imbalanced Classification for Botnet Detection in Internet of Things

The UNSW Bot-IoT dataset has a significant disparity between normal and attack traffic. “Synthetic Minority Over-sampling Technique (SMOTE)” (Chawla, 2002) with Support Vector Machine (SVM) is used to solve the imbalance in the data. The proposed technique is evaluated using classification metrics and ROC-AUC score. (Krishnan, et al. 2021)

Gaps: The ADASYN-SVM method results in higher true positives (TP) and true negatives (TN) values compared to the Synthetic Minority Over-sampling Technique – Support Vector Machines (SMOTE-SVM) method. This suggests that the Adaptive Synthetic Sampling – Support Vector Machines (ADASYN-SVM) model is better at detecting actual positives and ignoring actual negatives. The SMOTE-SVM method tends to produce more false negatives (FN) than the ADASYN-SVM method, which indicates that it struggles to detect actual negatives. Despite the imbalanced nature of the dataset, the ADASYN-SVM model achieves higher accuracy overall compared to the SMOTE-SVM method. (Ramadhan, et al. 2021)

2.14 IoT-botnet Detection using Fuzzy Rule Interpolation

Fuzzy Rule Interpolation (FRI) improved the accuracy and adaptability of network traffic classification systems, offering an alternative to traditional binary decision-making processes. FRI enhances the robustness of fuzzy systems, reducing system complexity and making intrusion detection more effective. The fuzzy nature of FRI helps in generating more realistic and comprehensive alerts, improving the overall response to potential threats. Fuzzy Rule Interpolation "offers a promising approach for enhancing network traffic classification" (Mouhammd, 2020) by capturing complex relationships and handling uncertainty.

Gaps: Fuzzy Rule Interpolation (FRI) can be expensive "computationally, especially when dealing with large" (Li, 2021) "rule base" and "high-dimensional input spaces." (Li, et al. 2021). The size and quality of the rule base play a crucial role in FRI's performance. A small, well-structured rule base is essential for effective interpolation. FRI can struggle with ambiguous or uncertain data, where multiple rules may be applicable, making it challenging to determine the correct interpolation result. FRI often relies on statistical patterns in the data without considering contextual information that might be relevant for accurate interpolation. FRI's performance can be sensitive to parameter tuning, such as rule weighting, threshold values, and aggregation methods, which may require careful adjustment. FRI is susceptible to noise or distortions in the data, leading to inaccurate interpolation results. FRI can introduce errors due to interpolation approximations, such as linear interpolation, which might not accurately represent the underlying relationships between rules. Traditional FRI methods often struggle with non-linear relationships between rules, where simple linear interpolations may not capture the underlying behavior. As the size of the rule base grows, FRI's

computational complexity increases exponentially, making it challenging to manage large-scale applications. Unlike classical rule-based systems, FRI often lacks guarantees about the accuracy or consistency of its results, making it less reliable in certain applications. FRI can struggle with missing values in the data, where interpolation methods may not be able to determine the correct value. FRI might not effectively manage heterogeneous data types or sources, which can lead to inconsistent results. (Li, et al. 2021)

2.15 Simulating attacks using MITRE Framework

Simulating attacks using MITRE CALDERA can be an effective approach to generate datasets for detecting novel exfiltration attacks. By emulating specific tactics, techniques, and procedures (TTPs) allows to create labelled datasets that reflect the complex and dynamic threats we face today. MITRE CALDERA allows you to simulate various stages of an attack, including initial compromise, lateral movement, data exfiltration, and post-exploitation activities. This enables you to generate datasets that mimic real-world scenarios, making it easier to train machine learning models for detecting novel exfiltration attacks. (Gjerstad, 2022)

Gaps: Creating an interaction between CTI (Continuous Threat Intelligence) and ISMS (Information Security Management System) using MITRE ATT&CK framework is a significant aspect of the proposed process. MITRE Caldera is an agent-based approach where security analysts execute threat hunting strategies directly within a virtual environment using pre-configured agents. In contrast, MITRE ATT&CK follows a process-based model that provides standardized techniques and methodologies for finding threats across networks without requiring manual or ad-hoc scripting of processes

on target systems. MITRE Caldera is lower level in terms of abstraction compared to MITRE ATT&CK as it requires security analysts to script out their own processes, whereas MITRE ATT&CK provides a standardized set of processes and techniques that can be executed without requiring manual scripting. Both frameworks can integrate with various other threat hunting and information security tools, but MITRE Caldera may require more customization than MITRE ATT&CK to achieve the same level of integration or automation in some cases. MITRE Caldera primarily focuses on data collection aspects while MITRE ATT&CK provides a unified approach for threat hunting and analysis across networks using standardized techniques, processes, and methodologies. MITRE ATT&CK offers more customization options than MITRE Caldera since it allows security analysts to script their own processes without requiring manual scripting or ad-hoc scripting of existing ones. This makes MITRE ATT&CK a bit more flexible compared to MITRE Caldera, as security analysts can tailor the framework according to their specific needs and workflows. MITRE ATT&CK framework can enable automated execution of their process chain, which allows for realistic “simulations to measure the effectiveness of countermeasures implemented and identify security gaps. MITRE ATT&CK framework enables automatic execution of their process chain and conduct proactive simulations.” (Mundt, et al. 2022). Utilizing the MITRE ATT&CK framework can automate parts of their process chain, making it more efficient.

2.16 Conclusion

Detecting novel data exfiltration attacks are typically known only after they have already been executed, making it difficult to detect them in real-time. While machine learning techniques allow to recognize patterns in data and predict potential threats, its

effectiveness “depends on the quality of the training data, the complexity of the attacks, and the performance” (Sabir, et al. 2020) metrics used. In summary, using machine learning for detecting novel data exfiltration attacks is not a silver bullet. A comprehensive approach that combines multiple techniques, including behavioral analysis, network monitoring, and anomaly detection, is likely to be more effective in detecting and mitigating these types of attacks.

Chapter 3—Methodology

3.1 Introduction

The dataset was developed by the “Information Security Group (ISG) at the University of New South Wales (UNSW). The dataset includes various attack categories such as OS and Service Scans, Distributed Denial of Service, and Data Exfiltration” (Koroniotis, 2019) attacks. Data Exfiltration can be conducted using various techniques through malware, phishing, and exploiting vulnerabilities which lead to unauthorized access to systems. Other attacks include Operating System (OS) and Service Scan which involves scanning the network to identify open ports, services, and operating systems, which can be exploited later. ”Denial of Service (DoS) and other attack variants (Distributed Denial of Service)” (Koroniotis, 2019) can target different network layers (TCP, HTTP).

3.2 Attack Dataset Generation

This section explains how different tools were used to perform various network attacks to generate the necessary network traffic dataset. This will give us insight into how these network attacks behave in an enterprise network.

Nmap was used to perform port scans. The first step in any probing attack is to identify the target(s) for further analysis. This could involve determining IP addresses, hostnames, or even subnets within an organization's network. In this case, a specific IP address was targeted. Once the target(s) have been identified, the appropriate scan type based on their goals and the level of stealth required was selected. A comprehensive scan (-A option) was used to gather information about the target machine. After choosing the

appropriate scan type, the attacker initiates the scan using nmap. This will perform a comprehensive scan of the target machine, with options such as -p to specify the port(s) to scan, -A for an aggressive scan that includes OS detection and script scanning, -oN to save the results to a file, and -v to increase verbosity. After the scan is complete, the results were analyzed to determine potential vulnerabilities, open ports, or other valuable information that can be exploited further. It was discovered that the target machine had an open port (presumably 80 for a web server) and was running Ubuntu Server.

"Nmap was used for network inventory, managing service upgrade schedules, and monitoring service uptime. Nmap's ability to quickly recognize all the devices on single or multiple networks, including servers, routers, switches, and mobile devices, made it possible to perform network scans to identify services running on a system, including HTTP, DNS, and other common services, with the capability to detect application versions with reasonable accuracy." (Koroniotis, 2019)

Nmap allowed the discovery of devices, assessment of vulnerabilities, and understanding of network structures as well as extending from simple port scanning to complex network mapping.

"Nmap could find information about the operating systems running on devices, providing details like OS versions, which aided in planning additional approaches during penetration testing. Nmap was also used to attack systems using existing scripts from the Nmap Scripting Engine. Stealth scanning is performed by sending a SYN packet and

analyzing the response to detect if the port were open, therefore a TCP connection could be opened.” (Koroniotis, 2019)

Nmap was used for network mapping, showing what types of devices were connected to the network and the scanned ports which allowed for the visualization of how servers, routers, switches, and other devices were connected and how they worked together. Port scanning with Nmap checked which ports were open and which were closed. Nmap could show if any of the connected machines used that app. OS Fingerprinting allows the discovery of several types of OS running on a device, including the brand and the patch level of the OS, as well as the estimated uptime of the endpoint. Lastly, Nmap was used for showing both the type and the location of machines connected to the network. (Koroniotis, 2019)

Meterpreter can establish a reverse TCP connection to allow the attacker to remotely access the system. Once connected, it enables attackers to navigate the file system, upload or download files, and exfiltrate sensitive information like entire directories. Using the SMB EternalBlue vulnerability allows to compromise a system and then deploying Meterpreter to extract data. Meterpreter allows attackers to elevate privileges to access sensitive areas or processes of the system. It includes a built-in keylogger to monitor and capture user input, such as passwords and other credentials. Captured keystrokes can be saved and sent to the attacker’s machine for further analysis.

Meterpreter commands such as 'sessions' were used for managing multiple sessions, while 'shell' allowed for direct interaction with the system's operating system. File manipulation was executed through commands like 'download', which enabled the retrieval of files from the target machine. The 'migrate' command was particularly useful

for maintaining persistence and evading detection by moving the session to different processes. For privilege escalation, 'getsystem' was the command of choice, granting full control over the system. Meterpreter's capabilities include scanning and discovering devices connected to the network for mapping out network infrastructure and identifying further targets for assessment. Meterpreter can run arbitrary code to access the file system, networking interfaces, and system registry. This can also be used after the initial compromise of a system to maintain control, gather further information, or propagate to other accessible systems. Meterpreter operates by injecting itself into an existing process on the target system, which allows it to evade detection by not creating new processes that could alert a system administrator or security software. Communication between the attacker and the Meterpreter payload is encrypted to avoid detection. Reverse_tcp shell allows it to connect back to the attacker's machine, which can be set up to listen for incoming connections from the compromised system. This reverse connection method is often used to traverse firewalls, as outbound connections from the target system are more likely to be allowed. Meterpreter can migrate between processes, upload and download files, capture screenshots and keystrokes, and execute system commands as well as escalate privileges and manipulate network connections. (Koroniotis, 2019)

Meterpreter was used in conjunction with nmap to identify potential targets with known vulnerabilities, exploit them, and gain a foothold on the targeted systems. First, nmap was used to scan target machines for open ports, operating system identification, and potential vulnerabilities. The scans were focused on Windows 7 and Ubuntu Server systems. This will perform a comprehensive scan of the target machine, with options such as -p to specify the port(s) to scan, -A for an aggressive scan that includes OS

detection and script scanning, -oN to save the results to a file, and -v to increase verbosity. After identifying potential targets using nmap, any known vulnerabilities can be found on those machines using the Metasploit Framework. The exploited vulnerabilities were specific to Windows 7 and Ubuntu Server systems. Metasploit's multi/browser/ms17_010_eol exploit to target a Windows 7 machine was used. RHOSTS option is then adjusted to specify the target IP address and execute the exploit using the run command. Once a successful exploitation occurs, Metasploit automatically establishes a Meterpreter session with the compromised machine. This allows interaction with the system remotely and exfiltrate data or perform data exfiltration. A reverse TCP connection can then be established on the attacker's machine and the Meterpreter payload was sent to the target machine. (Koroniotis, 2019)

For Denial-of-Service (DoS) attacks, the first step is to identify the target(s), which could be a specific IP address, hostname, or service running on a server. The Apache web server (running on port 80) was targeted. After identifying the target(s), the attacker chooses an appropriate method for their DoS attack based on available resources, goals, and the specific vulnerability of the targeted system. Apache Bench (ab) was used to generate a large amount of HTTP requests to overwhelm the server and simulate a DoS scenario. Apache Bench (ab) generates 10,000 HTTP requests with over 100 concurrent connections to the targeted web server. The number of requests and concurrent connections can be adjusted based on available resources and the desired level of impact. After executing the simulated DoS attack, the results can be analyzed to assess the

effectiveness of their attack and determine whether it has successfully overwhelmed the targeted server. (Koroniotis, 2019)

3.3 Dataset Composition

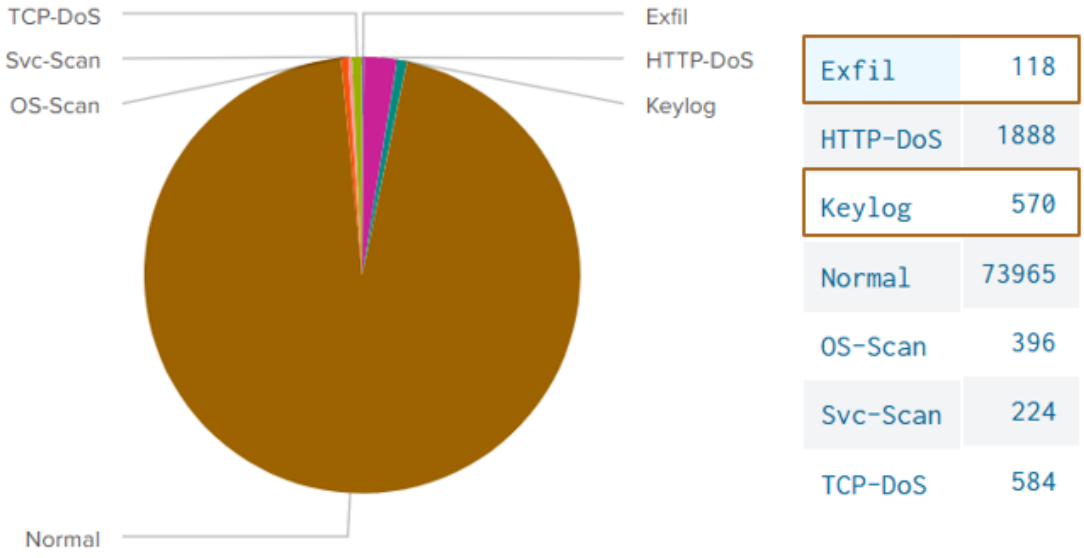

Figure 1 – Dataset Composition

Figure 1 shows the composition of the UNSW dataset. The pie chart visually represents the distribution of different attack types within a dataset. Normal Traffic dominates the chart, indicating that the majority of the dataset consists of benign, non-malicious network traffic. TCP-DoS represents TCP SYN flooding attacks, a common type of denial-of-service (DoS) attack. Svc-Scan refers to service scanning attacks, where an attacker scans open ports on a system to identify potential vulnerabilities. OS-Scan represents operating system fingerprinting attacks, where an attacker attempts to identify the operating system and version of a target system. Data-Exfil data exfiltration and Keylogger attacks, where an attacker attempts to steal sensitive data from a system. HTTP-DoS represents HTTP-based DoS attacks, which target web servers. The dataset exhibits a class imbalance, with normal traffic comprising the vast majority. This is a

common challenge in network intrusion detection datasets. The dataset includes a variety of attack types, which is important for training a robust intrusion detection system.

3.4 Dataset Features

Features	Definition	Data Type
Stime	Start time of the capture	Milliseconds
Dur	Duration of Transmission	Milliseconds
Rate	Rate of Transmission	Packets per second
Pkts	Packet Count	Integer
Bytes	Packet Length	Bytes
Spkts	Source Packet Count	Integer
Sbytes	Source Packet Length	Bytes
Dpkts	Destination Packet Count	Integer
Dbytes	Destination Packet Length	Bytes
Subcategory	Attack subcategory	Text

Table 1 – Dataset Features

Table 1 represents features specifically the start and end times of a transaction, which can be useful for identifying anomalies like rapid bursts of traffic. Duration (dur) represents the duration of a transaction, which can be helpful in detecting long-lived connections that might be associated with malicious activity. Other features capture information about the state of a transaction, including protocol types and flag settings. Analyzing these features can help identify unusual traffic patterns. IP Address and Port Information: saddr, sport, daddr, and dport: “These features provide information about the source and destination IP addresses and port numbers.” (Koroniotis, 2019). Analyzing these features can help detect anomalous traffic flows or port scanning activities. Traffic Volume Features such pkts, bytes, spkts, dpkts, sbytes, and dbytes capture information about the volume of traffic, including packet counts and byte counts in both directions. Analyzing these features can help detect unusual traffic spikes or large data transfers.

Rate-Based Features: rate, srate, and drate capture information about the rate of traffic, including packets per second. Analyzing these features can help identify rapid bursts of traffic or slow-moving attacks.

Packet Total Duration (dur) metric measures the total duration of all packets in a specific network capture or session. Long packet durations that might indicate heavy data transfer or encryption/decryption processes short packet durations that might indicate fast-paced communication protocols. Sudden changes in packet duration could signal the onset of a denial-of-service (DoS) attack.

Source-to-Destination Packet Count (spkts) metric measures the total number of packets sent from a specific source (e.g., an IP address) to a destination (e.g., another IP address). This provides insight into the volume and directionality of network communication which can be used to identify the most active sources and destinations on a network, detect anomalies or unusual patterns in packet transmission rates between specific source-destination pairs, and investigate potential malware infections by analyzing packet transmission rates from compromised hosts.

Destination-to-Source Packet Count (dpkts) metric measures the total number of packets sent from a specific destination (e.g., an IP address) back to a source (e.g., another IP address). This value provides insight into the volume and directionality of network communication. This value can be used to identify potential responses or acknowledgments from servers or services, detect anomalies or unusual patterns in packet transmission rates between specific destination-source pairs.

Source-to-Destination Bytes Count (sbytes) measures the total amount of data transmitted from a specific source (e.g., an IP address) to a destination (e.g., another IP

address). This value provides insight into the volume and directionality of network communication. This value can also be used to identify the most active sources and destinations on a network.

Destination-to-Source Byte Count (dbytes) metric measures the total amount of data transmitted from a specific destination (e.g., an IP address) back to a source (e.g., another IP address). This value provides insight into the volume and directionality of network communication. This value can be used to identify potential responses or acknowledgments from servers or services.

Rate represents the total number of packets per second. Rate can indicate unusual activity or potential attacks. “High packet rates indicate potential denial-of-service (DoS) or distributed denial-of-service (DDoS) attacks.” (Koroniotis, 2019). Unusual packet rates may indicate scanning or probing activities, which can be a precursor to an attack. Changes in packet rates can indicate a potential attack or unauthorized activity.

3.5 Methodology Diagram

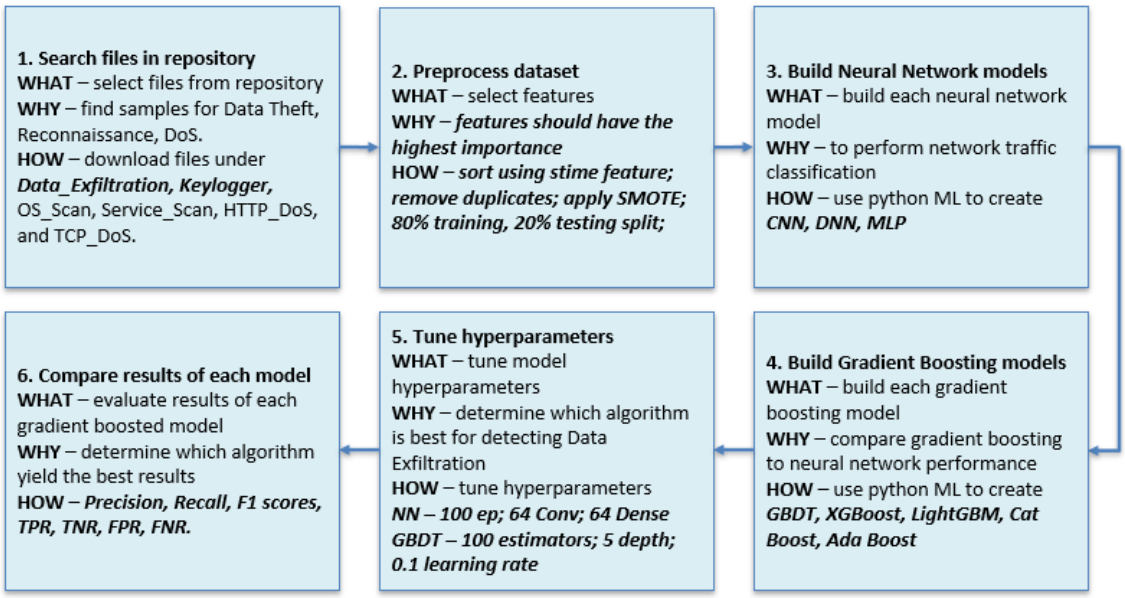

Figure 2 – Methodology Diagram

Figure 2 shows the overall process for detecting data exfiltration. This methodology focuses on building and comparing the performance of GBDTs and neural network models for detecting data exfiltration. In summary, Data Preparation collects and preprocesses the dataset, including handling class imbalance with SMOTE. Hyperparameter Tuning optimizes the hyperparameters of both neural network and GBDTs to be used to "evaluate the performance of these models using metrics specifically precision, recall, F1 score, and others." (Wikipedia)

Methodology Steps

Step 1. Search files in repository.

WHAT: Select files from the repository.

WHY: Find samples for Data Theft, Reconnaissance, and DoS.

HOW: Download files named Data_Exfiltration, OS_Scan, Service_Scan, HTTP_DoS, and TCP_DoS.

The first step is to identify and download specific files from a repository that represent different types of network intrusions. These files will likely contain network traffic data or logs that can be used to create machine learning model for detecting these types of network attacks. File Selection focuses on files related to Data Exfiltration, Keylogger, OS Scanning, Service Scanning, HTTP DoS, and TCP DoS attacks. Data Format: The files could be in various formats, such as PCAP (packet capture), CSV, or JSON. Data Quality ensures the selected files are of good quality, have accurate labels, and represent a diverse range of attack scenarios. Data quality and labeling are required in order to "ensure the accuracy and consistency of labels associated with the data." (Wikipedia) Inaccurate labels can negatively impact prediction performance. For Data Quantity and Diversity, we need a sufficient amount of diverse data which is essential for training a robust model. Lastly feature extraction, normalization, and handling missing values, may be necessary to "prepare the data for training the model." (Wikipedia)

Step 2. Preprocess dataset

WHAT: Select labels and features.

WHY: Select labels based on attack category and weights based on packet measurements.

HOW: Remove duplicates, sort using the stime feature, split data using 80% for training and 20% for testing, and apply SMOTE.

"Data Preprocessing is a critical step in machine learning when dealing with imbalanced datasets in order to prepare the data for effective model training and evaluation." (Wikipedia). Key steps involved are as follows: label selection using attack categories based on the downloaded files (Data_Exfiltration, Keylogger, OS_Scan, Service_Scan, HTTP_DoS, TCP_DoS), which correspond to these attack types. Normal

Traffic: A "Normal" label should be assigned to benign traffic. For Feature Selection, we look at Packet-Based Features. We extract features from each packet, such as: packet length and inter-arrival time, packet size and content (if applicable). Next, we look at Flow-Based Features. We aggregate packet-level features to create flow-level features, like number of packets, total bytes transferred, Source and destination byte counts. In the next step, we perform data cleaning by removing duplicates to eliminate redundant data points. Next step is to look into Data Splitting. For Train-Test Split, ”we divide the dataset into training and testing sets.” (Scikit Learn). Next step is to split training and testing using 80/20. The next step is Data Balancing using SMOTE. We address Class Imbalance by generating synthetic samples of several types of attacks. This helps the model learn from underrepresented classes and improve overall performance.

Step 3. Build Neural Network models.

WHAT: Build each neural network model.

WHY: Perform network attack classification.

HOW: Use Python ML to create each model.

The next step is to build multiple neural network models. First, the input layer ingests preprocessed network traffic data, matching the number of neurons to features. Second, multiple hidden layers learn complex patterns through tunable neurons and activation functions like ReLU or LeakyReLU. The output layer has neurons matching classes, using softmax activation for probability output. “Training involves categorical cross-entropy loss and Adam or SGD optimizers. The dataset is split into training and validation sets to monitor overfitting.” (Scikit Learn). Hyperparameters are adjusted for optimal performance. Finally, over-sampling techniques like SMOTE generate synthetic

minority class samples to address imbalance. This structured approach enables effective model training and evaluation for accurate network intrusion detection, optimizing performance through iterative refinement.

Step 4. Build GBDTs

WHAT: Build each GBDT.

WHY: Compare Gradient Boosting Machine to neural network performance.

HOW: Use Python ML to create each model.

We consider two popular libraries: Gradient Boosting and XGBoost, both known for their efficiency and performance. To optimize these models, we will "tune the learning rate, which controls the step size during training, the maximum depth of decision trees, the number of trees in the ensemble, and the subsample ratio," (Scikit Learn) which determines the fraction of samples used for training each tree. We will train both Gradient Boosting and XGBoost models on the preprocessed dataset, "using cross-validation to assess performance and fine-tune hyperparameters" for optimal results. (Scikit Learn)

Step 5. Tune hyperparameters

WHAT: Tune model hyperparameters.

WHY: Determine which algorithm is best for detecting Data Exfiltration.

HOW: Make changes to hyperparameters and run the code.

"Hyperparameter Tuning is a critical step in optimizing the performance of machine learning models." (Scikit Learn). It involves systematically "adjusting hyperparameters to find the best configuration" (Scikit Learn) for a given dataset and model architecture. To determine the best algorithm for detecting data exfiltration, we

will tune hyperparameters for CNN, DNN, MLP, GBDT, and XGBoost models. For neural networks (CNN, DNN, MLP), we will adjust “the number of layers, neurons per layer, learning rate, and optimizer.” (Scikit Learn). For GBDT and XGBoost, we will tune the “learning rate, maximum depth of trees, number of trees, and subsample ratio.” (Scikit Learn). By systematically modifying these hyperparameters and “evaluating the model's performance on a validation set,” (Scikit Learn) we can identify the optimal configuration for each algorithm and select the best-performing model for data exfiltration detection.

Step 6. Compare results of each model

WHAT: Evaluate results of each model.

WHY: Determine which algorithm yields the best results.

HOW: Calculate Precision, Recall, F1 scores, TPR, TNR, FPR, FNR.

To evaluate the performance of each model (CNN, DNN, MLP, GBDT, and XGBoost), we will calculate various metrics including “precision, recall, F1-score, true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), and false negative rate (FNR).” (Scikit Learn). These metrics will help us assess the “model's ability to correctly identify data exfiltration attempts while minimizing false positives and false negatives.” (Scikit Learn). By comparing the performance metrics of each model, we can determine the most effective algorithm for detecting data exfiltration. We compare the “proportion of positive identifications that were actually correct (Precision), the proportion of actual positive cases that were correctly identified (Recall), the harmonic means of precision and recall (F1-score). We also compare the proportion of actual positive cases that were correctly identified as positive (True Positive Rate TPR), the

proportion of actual negative cases that were incorrectly identified as positive (False Positive Rate FPR, the proportion of actual negative cases that were correctly identified as negative (True Negative Rate TNR), and the proportion of actual positive cases that were incorrectly identified as negative (False Negative Rate FNR).” (Scikit Learn). We then look at the overall correctness of the model's predictions. We analyze the performance of the model in different classes, especially minority classes, to identify areas for improvement and consider the trade-off between these metrics. A high FPR can lead to false alarms, while a high FNR can result in missed attacks. For Visualization Techniques, we use Confusion Matrix which provides a “visual representation of the model's performance, showing correct and incorrect predictions for each class.” (Wikipedia)

3.5.1 Methodology Step 1 – Data Files Selection

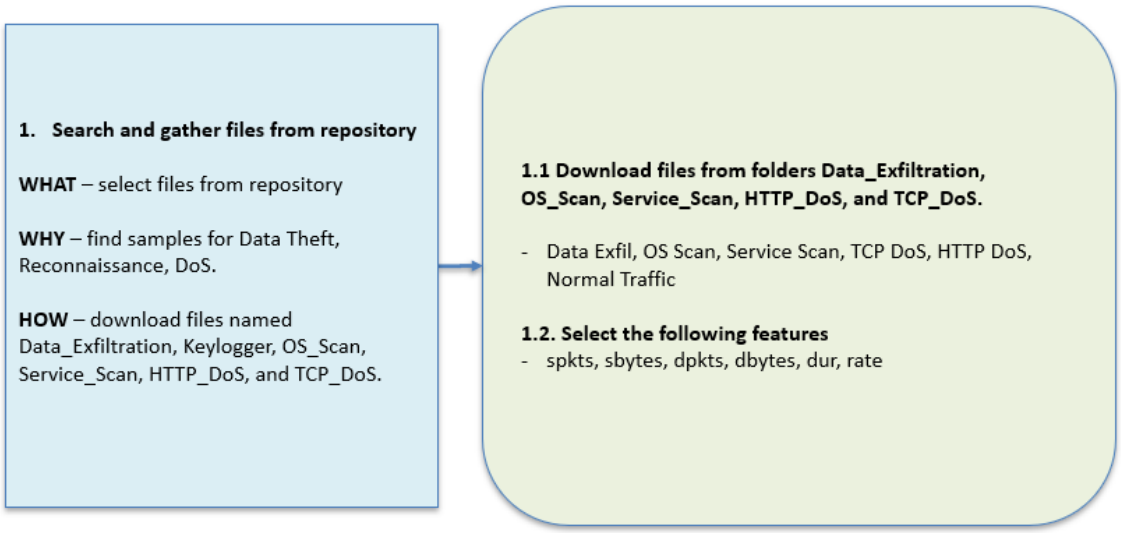

Figure 3 – Data Files Selection

Figure 3 shows data file selection and selecting a subset of features from the dataset that are deemed relevant for our detection model. This step focuses on the initial

data gathering and preparation phase. Files selection involves selecting the appropriate datasets that contain information related to data exfiltration, reconnaissance, and DoS attacks. Nest is acquiring the selected datasets from the repository. Feature Selection involves choosing a subset of relevant features (spkts, sbytes, dpkts, dbytes, dur, rate) that will be used for “training and testing the detection model.” (Scikit Learn).

Step 1 – Select files from repository:

WHAT: Select files from Data Theft, Reconnaissance, DoS folders.

WHY: To be used to build, train, test model.

HOW: Download files named Data_Exfiltration, Keylogger, OS_Scan, Service_Scan, HTTP_DoS, and TCP_DoS.

1.1 Download files named Data_Exfiltration, Keylogger, OS_Scan, Service_Scan, HTTP_DoS, and TCP_DoS.

1.2 Select the following features:

spkts, sbytes, dpkts, dbytes, dur, rate, pkts, bytes

We select the following features:

spkts: Number of packets sent.

sbytes: Number of bytes sent.

dpkts: Number of packets received.

dbytes: Number of bytes received.

dur: Duration of the connection

rate: Average rate of packets per second

pkts: Packet count

bytes: Packet Length

The selected features (spkts, sbytes, dpkts, dbytes, dur, rate, pkts, bytes) are fundamental for network traffic analysis and can be effective in detecting various network intrusions. Let us break down the significance of each feature: Basic Network Statistics: spkts: Number of packets sent. sbytes: Number of bytes sent. dpkts: Number of packets received. dbytes: Number of bytes received. These features provide insights into the volume and intensity of network traffic. Anomalies in these metrics can indicate potential attacks. Connection Duration and Rate: dur: Duration of the connection. rate: Average rate of packets per second. These features help identify unusual connection behaviors. For example, noticeably short or exceptionally long connections, or unusually high data rates, might be indicative of malicious activity. These features can be used as weights to assign importance to different data points during the training process. For instance, a data point with a high rate might be considered more significant.

These files contain data related to several types of cyberattacks and will be used for building, training, and testing the model. We will utilize the gathered files to develop and evaluate the machine learning model. Once the files are collected, they will be used in the following stages: Use the data to construct the architecture of the model. Train the model using data to learn patterns and features associated with several types of attacks. Evaluate the model to get class predictions. “Evaluate the model's performance using a separate test set of data to ensure its accuracy and effectiveness.” (Scikit Learn)

We search within the files for keywords related to the desired attack categories. Then we look for patterns like "Data-Exfil," “Keylogger,” "OS-Scan," "Service-Scan," "HTTP-DoS," "TCP-DoS," and "Normal" in the filenames. Datasets provide metadata files that describe the contents of each data file. We look for keywords or labels

indicating the presence of specific attack categories. If metadata is not available, we randomly sample files and examine their contents. Then we look for a column that indicates the attack category of each network flow. Next, we use text search techniques to identify files containing keywords related to the desired attack categories. We use a file search tool (find) on Linux or a similar tool on Windows to quickly locate files based on filenames or content. We write a script to automate the file search and extraction process. Then we use tools like head, tail, or less to quickly inspect the contents of files. Next is we extract the downloaded files to a specific directory. The dataset is typically organized into folders, each containing files for a specific attack category or normal traffic. Look for folders named Data Exfiltration, OS Scan, Service Scan, HTTP DoS, and TCP DoS. Within these folders, we will find CSV files containing the network traffic data. Use Python and Pandas to read and process these CSV files. Load the CSV files into Pandas DataFrames. Filter the DataFrames to select only the desired features and labels. Concatenate the filtered DataFrames from different attack categories into a single DataFrame.

3.5.2 Methodology Step 2 – Data Preprocessing

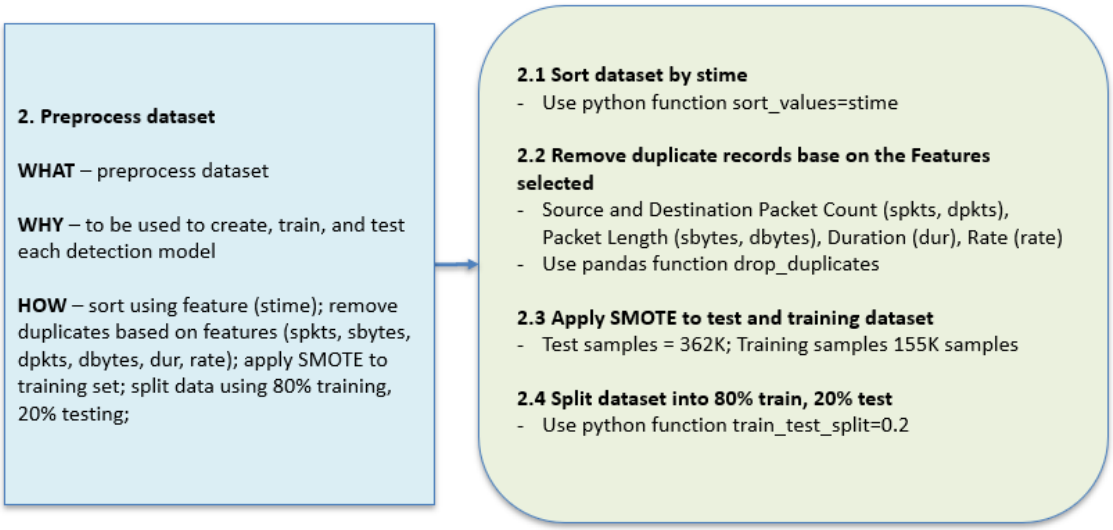

Figure 4 – Dataset Preprocessing

Figure 4 focuses on the data preprocessing phase to ensure data quality and suitability for model training and testing. Sorting the dataset by time (stime) helps in maintaining the temporal order of events. Removing duplicate records based on the selected features helps to ensure data integrity and prevent overfitting. “Splitting the dataset into training and testing sets” (Scikit Learn) is essential for model training and evaluation to see if the model is able to generalize. An 80/20 split is a common practice. We apply SMOTE to the training to address the class imbalance problem. The preprocessed dataset will be used to create, train, and test each detection model. Sort using feature (stime); remove duplicates based on features (spkts, sbytes, dpkts, dbytes, dur, rate); split data using 80% training, 20% testing; apply SMOTE to training set. Step 1 – sorts the dataset using feature stime. The data will be sorted based on the stime (start time) column.

Sorting the dataset by stime (start time) is a critical preprocessing step for network intrusion detection. It allows us to: Identify Sequential Patterns: By ordering the data by time, we can identify sequential patterns in network traffic that may indicate

attacks. For example, a series of rapid scans or a sudden burst of traffic could be signs of a potential intrusion.

Step 2 – removes duplicates based on features (spkts, sbytes, dpkts, dbytes, dur, rate). Duplicates will be removed by considering the combination of the following features:

spkts: number of packets sent.

sbytes: total bytes sent.

dpkts: number of packets received.

dbytes: total bytes received.

dur: duration of the connection

rate: network traffic rate

pkts: packet count

bytes: pack length

Any duplicate rows will be removed, leaving only unique combinations of these features. Removing duplicate data points is a crucial step in data preprocessing for machine learning models. Duplicate data can lead to biased models and negatively impact their performance. Implementation Considerations: To effectively remove duplicates based on the specified features, we can follow these steps: Feature Selection: Identify the key features that define a unique data point. In this case, the combination of spkts, sbytes, dpkts, dbytes, dur, and rate is used. Data Grouping: Group the data by these key features. Duplicate Removal: Within each group, identify and remove duplicate data points. This can be done by checking for identical values in all the key features. In some cases, near-duplicate data points may exist, where the values are remarkably similar but not identical. Techniques like clustering or similar measures can be used to identify and handle these

cases. “It is important to ensure that the data is clean and accurate.” (Sabir, 2021). Errors in data entry or measurement can lead to incorrect duplicate identification. Data Imbalance: Removing duplicates might exacerbate class imbalance, especially for minority classes.

Step 3 – apply SMOTE to the training set to address class imbalance issues if present. SMOTE creates synthetic samples by “interpolating between existing minority class instances and their nearest neighbors.” (Chawla, 2002). “Training set consists of 80% of the original data Sorted and deduplicated based on stime and features (spkts, sbytes, dpkts, dbytes, dur, rate). Testing set consists of 20% of the original data.” (Scikit Learn). Sorted and deduplicated based on stime and features (spkts, sbytes, dpkts, dbytes, dur, rate, pkts, bytes). The test dataset is modified with SMOTE.

Step 4 – split the data using 80% training, 20% testing. The sorted and deduplicated data will be split into two parts:

Training set (80%) : “used to train the model” (Scikit Learn)

Testing set (20%) : “used to evaluate the performance of the trained model” (Scikit Learn)

“Splitting the dataset into training and testing sets is a crucial step in machine learning. By dividing the data, we can train the model in one part and evaluate its performance on unseen data. Splitting ensures that the split is random to avoid bias in the dataset. The split should maintain the original class distribution in both the training and testing sets to accurately assess the model's performance on different classes. Data Imbalance: If the dataset is imbalanced, consider techniques like stratified sampling to ensure that the distribution of classes is preserved in both sets.” (Scikit Learn)

In summary, sorting by stime will maintain the temporal order of the data. This is crucial for time-series analysis and models that consider the sequential nature of network traffic. We identify and remove duplicate data points. Duplicates are identified based on the selected features: spkts, sbytes, dpkts, dbytes, dur, rate, pkts, and bytes. If two rows have identical values for these features, one is removed. We divide the dataset into two subsets: “Training Set which consists 80% of the data, used to train the detection model. Testing Set which consists of 20% of the data,” (Scikit Learn) used to evaluate the detection model's performance.

3.5.3 Methodology Step 3 – Build Neural Network models.

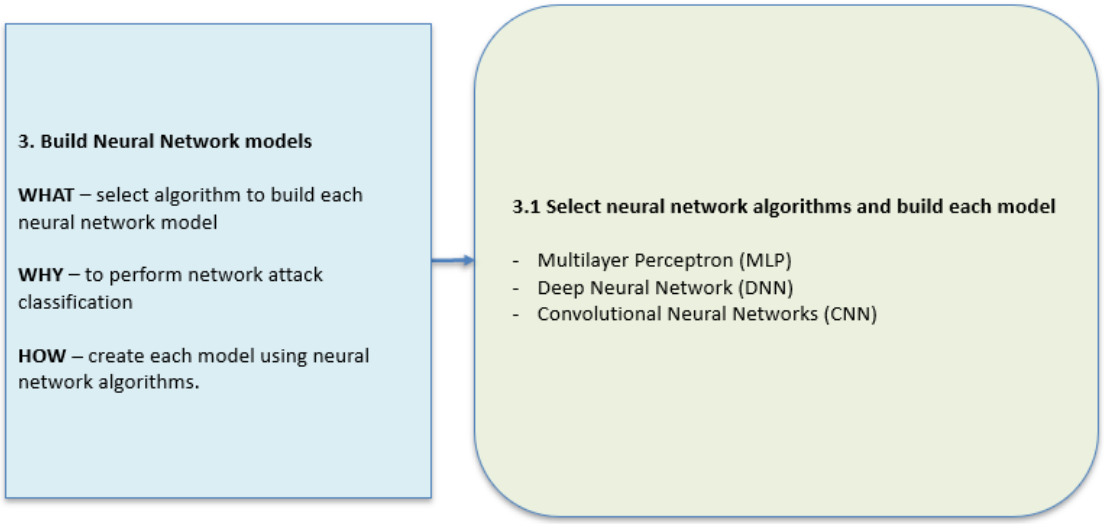

Figure 5 – Build Neural Network Models

Figure 5 focuses on the construction of neural network models. “Neural network architecture involves defining the layers, activation functions, and optimization algorithms. Training the model involves incrementally adjusting the model's parameters to minimize the loss function and improve performance.” (Hinton, et al. 1986)

“Multilayer Perceptron (MLP) hidden layer parameter specifies the number of neurons in the hidden layer. We have one hidden layer with thirty-two neurons. The

Rectified Linear Unit (ReLU) introduces non-linearity to the model, allowing it to learn complex patterns. The Adam optimizer adapts the learning rate for each parameter and is used to update the model's weights during training. Max_iter sets the maximum number of iterations for the optimization algorithm. Input Layer receives the input features. Hidden Layer processes the input features using the specified activation function (ReLU). Output Layer produces the final output, which is a probability distribution over the different classes." (Scikit Learn)

"Deep Neural Networks (DNN) Input Layer receives the input data. Hidden Layers process the inputs received from the input layer." (Dolhopolov, 2024). DNNs output layer produces the output. Each layer consists of neurons that process data using weights and biases. Activation functions are applied to the neurons to transform the input data into useful representations. "ReLU (Rectified Linear Unit), Sigmoid, and Tanh to introduce non-linearity into the network, allowing it to model complex relationships." (Dolhopolov, 2024). "Backpropagation involves adjusting weights based on the error between predicted and actual outputs. Optimization algorithms such as Stochastic Gradient Descent (SGD) or Adam are used to minimize the error. Overfitting occurs when a model performs well on training data but poorly on unseen data. Dropout, L2 regularization, and data augmentation are used to prevent overfitting." (Hinton, et al. 1986). "Sequential Model uses a sequential stack of layers. Dense Layers are fully connected (dense) layers. In the first layer, sixty-four neurons are built with ReLU activation. In the second layer, thirty-two neurons are built with ReLU activation. The output layer consists of six neurons with softmax activation for multi-class classification. Input Shape: The model expects an input with six features. Loss Function: Sparse

categorical crossentropy - suitable for multi-class classification with sparse labels. Adam optimizer is used for training neural networks. For Metrics, Accuracy - measures the percentage of correct predictions. Epochs = 88 are used for the entire training data which is passed through the network. Batch Size = 32 is the number of samples used to update the model's weights in each iteration.” (Scikit Learn). Verbose = 1 prints training progress information during each epoch.

Next, we build a “Convolutional Neural Network model using the Keras library. Sequential() creates a new instance of the Keras sequential API, which is used to define the architecture of the neural network. Conv1D(64, kernel_size=3, activation='relu', input_shape=(6, 1)) adds a convolutional layer (Conv1D) with 64 filters to the model. The kernel_size parameter specifies the size of the convolutional filter (in this case, 3). The activation parameter specifies the activation function used for the convolutional layer (ReLU in this case). The input_shape parameter specifies the shape of the input data, which is a 1D array with six elements. MaxPooling1D(pool_size=2) adds a max pooling layer (MaxPooling1D) to the model. The pool_size parameter specifies the size of the pool (in this case, 2). Conv1D(32, kernel_size=3, activation='relu') adds another convolutional layer with 32 filters to the model. The rest of the parameters are the same as in step 2. MaxPooling1D(pool_size=2) adds another max pooling layer to the model, just like in step 3. Flatten() flattens the output of the convolutional and pooling layers into a 1D array. Dense(128, activation='relu') adds a dense (fully connected) layer with 128 units to the model. The activation parameter specifies the activation function used for this layer (ReLU in this Dense(2, activation='softmax') adds another dense layer with 2 units to the model. The activation parameter specifies the activation function used for this layer

(softmax in this case), which is typically used for output layers that need to predict probabilities." (Scikit Learn). The resulting CNN model has an input shape of (6,1), meaning it expects a 1D array with six elements as input. (Scikit Learn)

3.5.4 Methodology Step 4 – Build GBDT models.

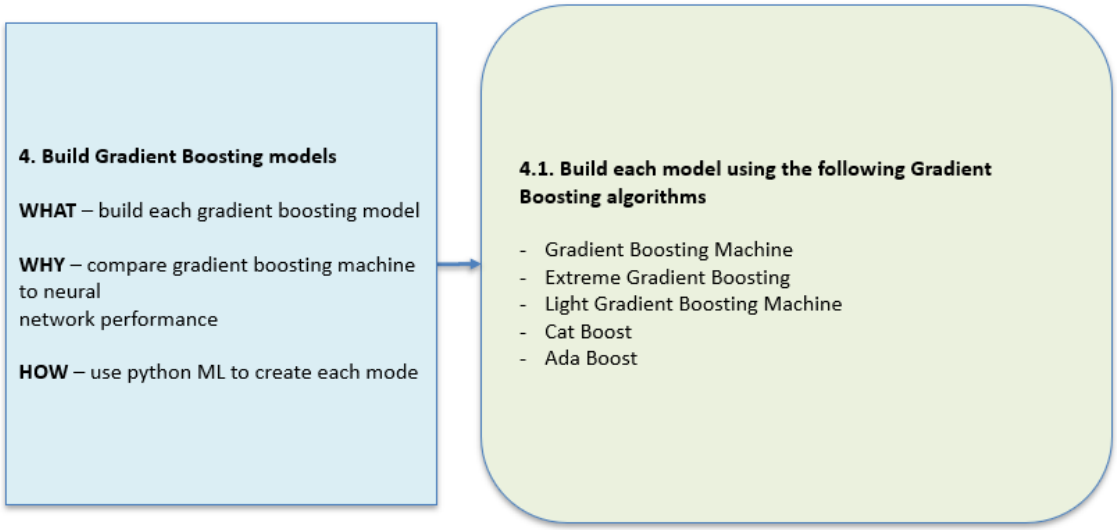

Figure 6 – Build GBDT models.

Figure 6 provides an alternative approach to the neural network-based models. Choosing the right "Gradient Boosted algorithm can significantly impact the model's performance." (Friedman, et al, 2002). Different algorithms have different strengths and weaknesses. Implementing the Gradient Boosted algorithm involves specifying the hyperparameters and configuring the training process. We build each of the following Gradient Boosted algorithms:

- Gradient Boosting Machine
- Extreme Gradient Boosting (XGBoost)
- Light Gradient Boosting Machine (LightGBM)
- Cat Boosting Machine
- Ada Boosting Machine

Training the GBDT involves iteratively building an ensemble of models, each focusing on correcting the errors of the previous ones. We use Python ML to create each GBDT. By following these steps, the GBDTs are built and trained. In building GBDTs, we compare Gradient Boosting Machine to neural network performance.

3.5.5 Methodology Step 5 – Apply SMOTE and Tune Hyperparameters

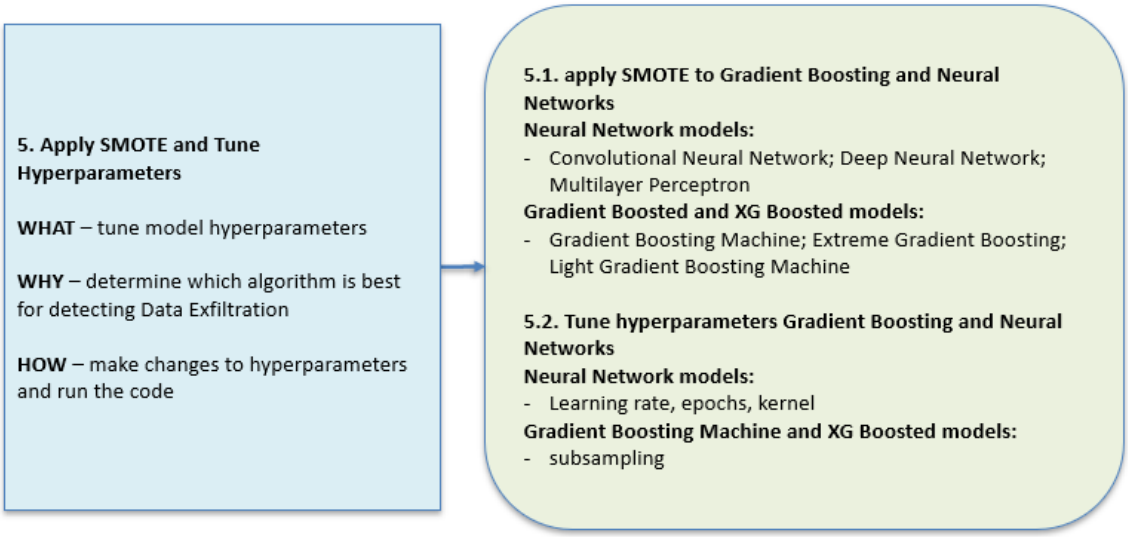

Figure 7 – Apply SMOTE and Tune Hyperparameters

Figure 7 focuses on fine-tuning the models and addressing class imbalance. We address the class imbalance in the dataset by applying "Synthetic Minority Oversampling Technique (SMOTE) to ensure that the models are not biased towards the majority class." (Chawla, 2002). Tuning the hyperparameters of both neural network and GBDTs to improve their performance. This involves adjusting parameters like learning rate, epochs, kernel size, and subsampling rate. Applying SMOTE addresses class imbalance is crucial for building robust models that can accurately detect minority classes like data exfiltration. "Fine-tuning hyperparameters can significantly impact the model's performance and generalization capability." (Scikit Learn)

3.5.6 Methodology Step 6 – Compare prediction results for each model.

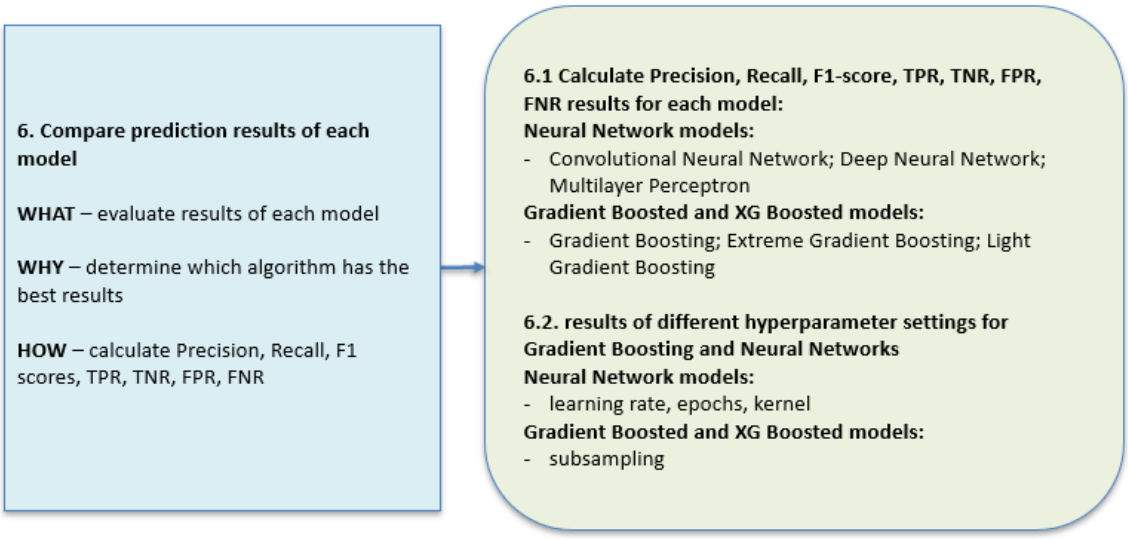

Figure 8 – Compare prediction results from each model.

Figure 8 compares prediction results from each model which is critical for selecting the best-performing model and understanding its strengths and limitations. By comparing the performance metrics obtained with different hyperparameter settings, it is possible to identify the optimal configuration for each model. Calculating these metrics helps in understanding the strengths and weaknesses of each model and comparing their overall performance.

"To determine the ratio of correctly predicted positive observations to all actual positives – True Positive Rate (TPR). First, we identify the number of positive instances correctly predicted by the model – True Positives (TP). Next, we identify the number of positive instances incorrectly predicted as negative by the model – False Negatives (FN). Then we divide the number of true positives by the sum of true positives and false negatives." (Jayaswal, 2020)

Formula:

$$\mathrm{TPR} = \frac{TP}{TP + FN}$$

Equation 1 – True Positive Rate

Next we calculate "the ratio of correctly predicted negative observations to all actual negatives – True Negative Rate (TNR). To calculate TNR, identify the number of negative instances correctly predicted by the model - True Negatives (TN). Next we identify the number of negative instances incorrectly predicted as positive by the model - False Positives (FP), then divide the number of true negatives by the sum of true negatives and false positives." (Jayaswal, 2020)

Formula:

$$\mathrm{TNR} = \frac{TN}{TN + FP}$$

Equation 2 – True Negative Rate

Next, we calculate "the ratio of incorrectly predicted positive observations to all actual negatives – False Positive Rate (FPR). To calculate FPR, we identify the number of negative instances incorrectly predicted as positive by the model - False Positives (FP). Then we identify the number of negative instances correctly predicted by the model - True Negatives (TN) then divide the number of false positives by the sum of false positives and true negatives." (Jayaswal, 2020)

Formula:

$$\mathrm{FPR} = \frac{FP}{FP + TN}$$

Equation 3 – False Positive Rate

Next, we calculate “the ratio of incorrectly predicted negative observations to all actual positives – False Negative Rate (FNR). Low FNR is important when false negatives are costly. To calculate FNR, identify the number of positive instances incorrectly predicted as negative by the model – False Negatives (FN). Next, we Identify the number of positive instances correctly predicted by the model - True Positives (TP) then divide the number of false negatives by the sum of false negatives and true positives.” (Jayaswal, 2020)

Formula:

$$\text{FNR} = \frac{FN}{FN + TP}$$

Equation 4 – False Negative Rate

Next, we calculate “the ratio of correctly predicted positive observations to the total predicted positives (ie. Precision). High precision is crucial when false positives are costly. We identify the number of positive instances correctly predicted by the model – True Positives (TP). We identify the number of negative instances incorrectly predicted as positive by the model – False Positives (FP) then divide the number of true positives by the sum of true positives and false positives.” (Jayaswal, 2020)

Formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Equation 5 – Precision

We calculate “the ratio of correctly predicted positive observations to all observations in the actual class (ie. Recall). High recall is important when false negatives are costly. We identify the number of positive instances correctly predicted by the model

– True Positives (TP). We identify the number of positive instances incorrectly predicted as negative by the model – False Negatives (FN) then divide the number of true positives by the sum of true positives and false negatives." (Jayaswal, 2020)

Formula:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Equation 6 – Recall

Lastly, we calculate "the harmonic mean of Precision and Recall (ie. F1 Score), providing a balance between the two. It is useful when both precision and recall are important, such as in information retrieval systems where both relevant results and few irrelevant results are desired. First, calculate Precision and Recall using the steps above. Use the formula to find the harmonic mean of Precision and Recall." (Jayaswal, 2020)

Formula:

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

Equation 7 – F1 Score

"Precision is significant when false positives are costly while Recall is significant when false negatives are costly. F1 Score is useful when both balances precision and recall are important." (Jayaswal, 2020). TPR (Recall) is important for understanding sensitivity. TNR (Specificity) is important for understanding how well the model identifies negatives. FPR is important to minimize in scenarios where false positives are costly. FNR is important to minimize in scenarios where false negatives are costly.

3.6 Conclusion

We built the neural network models as well as GBDT models and applied SMOTE. Neural Networks benefited from SMOTE and performed significantly better than using SMOTE. Our methodology focuses on applying SMOTE to improve CDD, DNN, and MLP in detecting data exfiltration. In summary, Data Preparation collects and preprocesses the dataset, and handling class imbalance with SMOTE. We built CNN, DNN, and MLP models as well as GBDTs. We optimize the hyperparameters of both neural network and GBDTs. We evaluated the performance of both types of models using precision, recall, F1 score, and looked at TPR, FPR, TNR, and FNR.

Chapter 4—Results

4.1 Introduction

We compare the results from traditional deep learning models, specifically "Multilayer Perceptrons (MLP), Deep Neural Networks (DNN)," (Dolhopolov, 2024) and "Convolutional Neural Networks (CNN)." (Xin, 2023)

4.2 Convolutional Neural Networks (CNN) Results

Figure 9 – CNN confusion matrix.

Figure 9 shows key observations from the Confusion Matrices show improved performance using SMOTE. This shows a "significant improvement in the model's ability to correctly classify instances, specifically for the minority classes." (Chawla, 2002). The original dataset might have had a class imbalance problem, as evident by the higher number of instances in the "Normal Traffic" class. SMOTE helped address this issue by

oversampling the minority classes. The improvement is particularly noticeable for classes like "Data Exfil", which were underrepresented in the original dataset.

Post-SMOTE application shows a significant improvement in the model's ability to correctly classify instances, specifically for the minority classes. The original dataset had a class imbalance problem, as evident by the higher number of instances in the "Normal Traffic" class. SMOTE helped address this issue by oversampling the minority classes. The improvement is particularly noticeable to "Data Exfil", which were underrepresented in the original dataset. Without using SMOTE, we can see that the model has an exceptionally low recall for this class. This means that CNN is not able to correctly identify many of the actual data exfiltration instances. However, after applying SMOTE, the recall for the "Data Exfil" class has significantly improved. This indicates that CNN is now better able to detect data exfiltration attacks.

4.3 Deep Neural Networks (DNN) Results

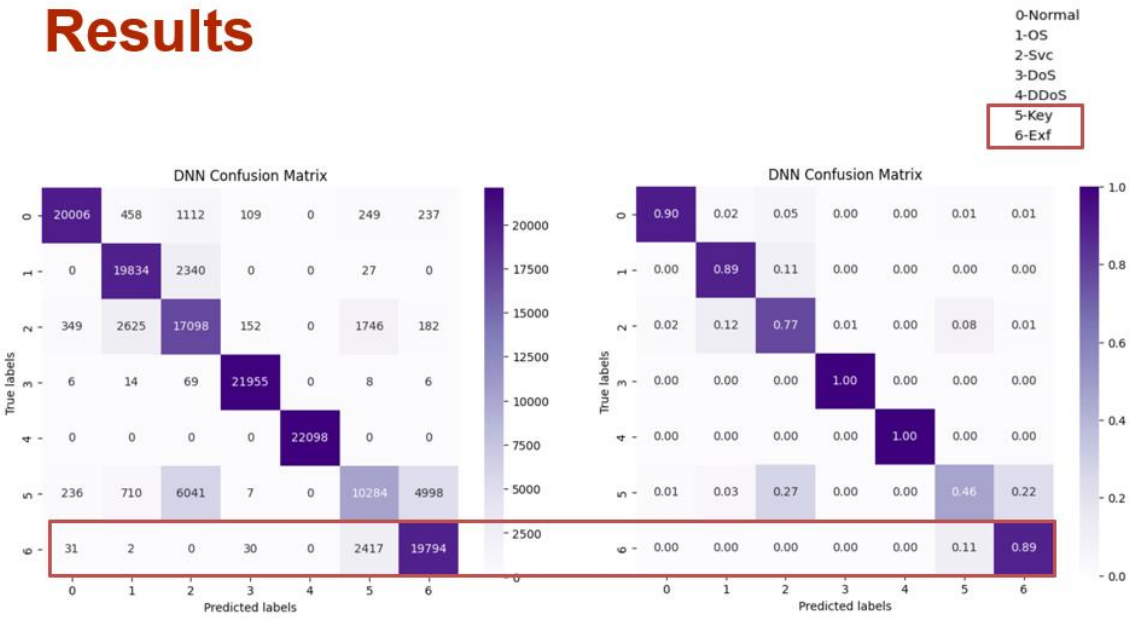

Figure 10 – DNN confusion matrix.

Figure 10 show key observations from the Confusion Matrices. The original dataset had a significant class imbalance, with "Normal Traffic" being the majority class and "Data Exfil" being minority classes. The application of SMOTE has led to a noticeable improvement in DNN's performance, particularly for the minority classes. We note specific improvements with SMOTE. There is an increased Recall for Minority Classes. SMOTE helps DNN to learn better from the underrepresented minority classes. This is evident in the improved recall values for classes like "Data Exfil". There is improved Overall Accuracy. The overall accuracy of DNN has also increased with the application of SMOTE. This is because DNN is now better equipped to correctly classify instances from all classes. The highlighted portion shows a significant improvement in DNN's performance for this class after applying SMOTE. Specifically, the recall value for this class has increased, indicating that the model is now better at identifying actual instances of data exfiltration. This improvement is due to "SMOTE addressing the class imbalance issue in the dataset. By generating synthetic samples of the minority class," (Chawla, 2002), SMOTE helps DNN learn the patterns of this class more effectively, leading to better classification performance.

4.4 Multilayer Perceptron (MLP) Results

Evaluation of Multilayer Perceptron (MLP) results using SMOTE.

Figure 11 – MLP confusion matrix.

Figure 11 shows increased Recall for Minority Classes. “SMOTE helps the model to learn better” (Chawla, 2002) from the underrepresented minority classes. This is evident in the improved recall values for classes like "Data Exfil" in the second confusion matrix. The overall accuracy of the model has also increased with the application of SMOTE. This is because MLP is now better equipped to correctly classify instances from all classes. “SMOTE helps the model to learn better” (Chawla, 2002) from the underrepresented minority classes. This is evident in the improved recall values for classes like Data Exfil in the confusion matrix. The overall accuracy of MLP has also increased with the application of SMOTE however, MLP does not seem to be equipped for network traffic classification.

4.5 Gradient Boosted Decision Tree Results

Evaluation of Gradient Boosted Decision Tree results using SMOTE.

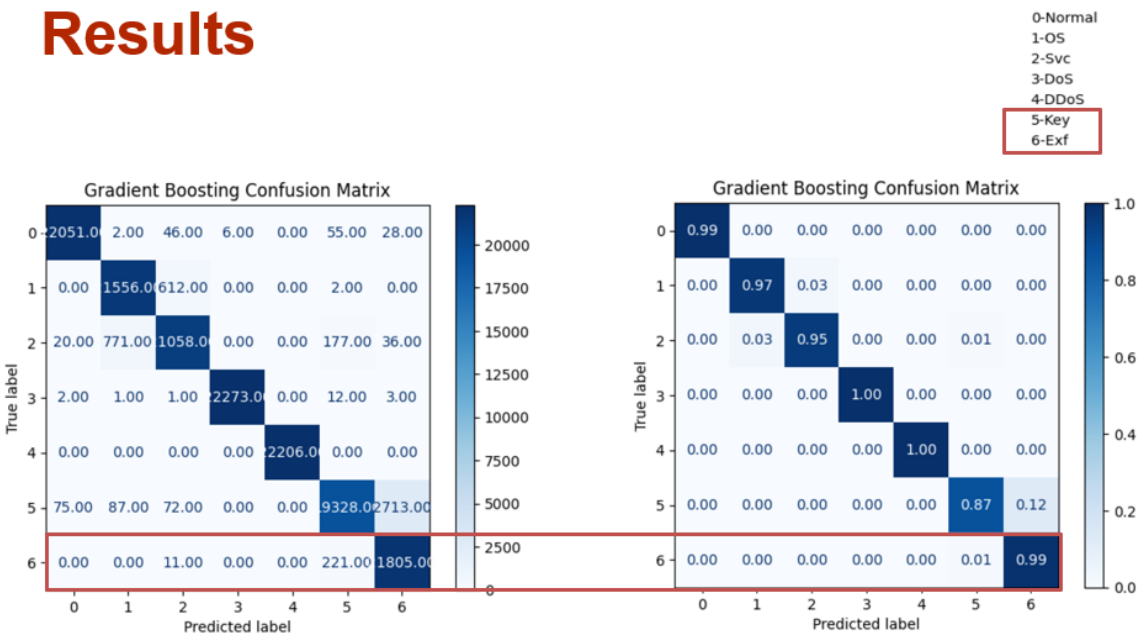

Figure 12 – GBDT confusion matrix.

Figure 12 show a slight to moderate increase Recall for Minority Classes. "SMOTE helps the model to learn better from the underrepresented minority classes." (Chawla, 2002). This is evident in the improved recall values for classes like "Data Exfil" and "Keylogger" in the second confusion matrix. There is also improved overall Accuracy. The overall accuracy of the model has also increased with the application of SMOTE. This is because GBDT is now better equipped to correctly classify instances from all classes. The original dataset had a significant class imbalance, with "Normal Traffic" being the majority class and "Data Exfil" and "Keylogger" being minority classes. The application of SMOTE has led to a noticeable improvement in the model's performance, particularly for the minority classes. "Precision shows the proportion of positive identifications that were actually correct. Recall shows the proportion of actual positive cases that were correctly identified. F1-score shows the harmonic mean of

precision and recall, providing a balanced measure of performance. Support shows the number of samples of the true class in the dataset.” (Jayaswal, 2020). GBDT shows specific improvements with SMOTE. There is an increased recall for Minority Classes. “SMOTE helps the model to learn better from the underrepresented minority classes.” (Chawla, 2002). This is evident in the improved recall values for classes like "Data Exfil" and "Keylogger" in the second classification report. There is improved overall Accuracy. The overall accuracy of GBDT has also increased with the application of SMOTE. This is because the model is now better equipped to correctly classify instances from all classes.

The impact of SMOTE helped improve the performance of classification models, especially for minority classes. Comparing the models' performance with and without SMOTE based on the confusion matrices provided, we can observe the following: DNN and CNN both benefit from SMOTE. The increased recall for minority classes indicates that SMOTE helps the models learn better from underrepresented data. While SMOTE can also improve the performance of GBDTs, the magnitude of improvement might vary “depending on the specific dataset and model configuration.” (Friedmann, 2001). Some Key Factors Influencing Model Performance: Data Quality and Quantity of large and diverse dataset is crucial for training effective models. “Fine-tuning hyperparameters like learning rate, batch size, and number of epochs can optimize model performance. Model Architecture depends on the specific problem and dataset characteristics.” (Scikit-Learn). In Class Imbalance Handling, techniques like SMOTE can help address class imbalance and improve model performance. The most significant improvement is observed for the "Data-Exfil" class. The TPR has increased significantly after “applying SMOTE,

indicating that the model is better able to correctly identify instances of this minority class.” (Chawla, 2002). Minimal Impact on Majority Class: The performance of the majority class ("Normal") remains consistently high, both with and without SMOTE. Overall Performance: While “the overall performance of the detection model is already on par,” (Chawla, 2002), SMOTE has helped to further improve the classification accuracy, especially for the minority classes.

4.6 XGBoost Results

Evaluation of XGBoost results.

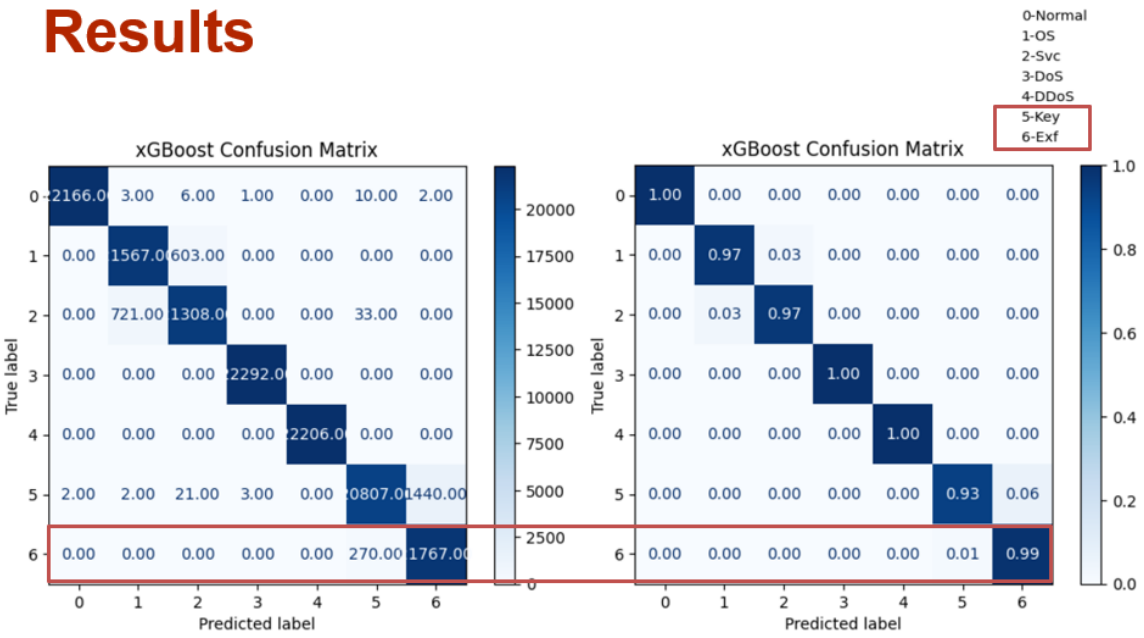

Figure 13 – XGBoost confusion matrix.

Figure 13 – XGBoost confusion matrix results show increased Recall for Minority Classes. “SMOTE helps the model to learn better from the underrepresented minority classes.” (Chawla, 2002). This is evident in the improved recall values for classes like "Data Exfil" and "Keylogger" in the second confusion matrix. There is also improved overall Accuracy: The overall accuracy of XGBoost has also increased with the

application of SMOTE. This is because XGBoost is now better equipped to correctly classify instances from all classes. This indicates that the model is highly accurate in classifying network traffic into the correct categories.” (Alsarhan, 2023). The high accuracy and precision for the majority class (class 0, "Normal Traffic") suggests that the model effectively distinguishes normal traffic from anomalous activity. However, the lower recall for minority classes (OS Scan and Service Scan) indicates that the model might struggle to correctly identify instances of these classes. SMOTE addressed the class imbalance and has improved the model's ability to detect these minority classes. The significant improvement in the recall for Keylogger, in particular, highlights the effectiveness of “SMOTE in addressing class imbalance. By generating synthetic samples of the minority class,” (Chawla, 2002). SMOTE helps the model to learn better from underrepresented classes. Key observations from the SMOTE comparison shows minimal impact on Majority Class: Both models perform exceptionally well on the majority class ("Normal"), with exceptionally low FPR and high TPR, regardless of SMOTE application. Limited Impact of SMOTE on Minority Class, in this specific case, SMOTE seems to have minimal impact on the performance of the "Data-Exfil" class. Both models achieve similar TPR and FNR values.

4.7 Overall Classification Results

Model	Precision	Recall	F1-score	TPR	FPR	TNR	FNR
GBDT	**0.89**	**0.99**	**0.94**	**0.989**	**0.021**	**0.979**	**0.011**
XGBoost	**0.94**	**0.99**	**0.96**	**0.988**	**0.011**	**0.989**	**0.012**
LightGBM	**0.93**	**0.99**	**0.96**	**0.990**	**0.012**	**0.988**	**0.012**
CatBoost	**0.93**	**0.99**	**0.96**	**0.988**	**0.012**	**0.988**	**0.012**
AdaBoost	**0.97**	**0.98**	**0.98**	**0.980**	**0.004**	**0.996**	**0.020**
CNN	0.76	0.95	0.85	0.949	0.049	0.951	0.051
DNN	0.78	0.89	0.83	0.889	0.041	0.959	0.111
MLP	0.61	0.88	0.72	0.880	0.096	0.904	0.120

Table 2 – Data Exfiltration Detection Performance Comparison

Table 2 shows the Macro Averages of each algorithm. We use Macro Averages for the following reasons: “Macro averages treat all classes equally, regardless of their size. This is important to ensure that the model performs well across all classes, not just the majority class. By averaging the metrics (precision, recall, F1-score) for each class, macro averages provide a balanced view of the model's performance. This is particularly useful when the dataset has classes of significantly different sizes. Macro averages can highlight weaknesses in the model's performance for minority classes,” (Jayaswal, 2020). which might be overlooked if only overall accuracy is considered. Overall, Classification Performance of CNN performs well for Normal, HTTP DoS, and TCP DoS classes but struggles with Keylogger, OS Scan, and Service Scan classes. DNN shows impressive performance across all classes, especially Data Exfil, HTTP DoS, and TCP DoS. GBDT also performs well across all classes, with slightly lower TPR for Data Exfil compared to XGBoost. XGBoost demonstrates the best performance for all classes, with high TPR and low FPR. GBDT, XGBoost, LightGBM, CatBoost, and AdaBoost are the top-performing

models. They have high TPR and TNR values, indicating their ability to correctly classify both positive and negative cases. Their FPR and FNR values are also relatively low, suggesting fewer incorrect classifications. CNN performs reasonably well, but its FPR is slightly higher compared to the top-performing models. This means it might incorrectly classify more negative cases as positive. DNN has the lowest TPR and TNR values, indicating it struggles to correctly identify both positive and negative cases. Its FPR is also relatively high, suggesting a higher rate of false positives. Data Exfiltration and Keylogger Class on both GBDT and XGBoost models benefit from SMOTE, especially for the Data Exfiltration class. The TPR for this class has increased significantly after applying SMOTE, indicating better detection of data exfiltration attempts. Consistent Performance for Majority Class: Both models perform well for the majority class ("Normal"), with high TPR and low FPR, both with and without SMOTE.

4.8 Conclusion

Overall Performance of GBDT models show strong performance, with minimal differences in the FPR and TNR values. The improvement in TPR for the minority classes, particularly "Data-Exfil," is noteworthy. A Comparison between GBDT shows a significant improvement in the TPR for the "Data-Exfil" class after applying SMOTE. This suggests that GBDT is more sensitive to class imbalance and benefits more from techniques like SMOTE.

Chapter 5—Discussion and Conclusions

5.1 Discussion

Hypotheses 1: Feature Selection – Gradient Boosted Decision Trees leverages packet header features for detecting Data Exfiltration attacks. Packet headers contain structured fields that are directly suitable for GBDTs, minimizing the need for complex preprocessing. Compared to deep packet inspection which requires parsing payloads, analyzing headers is computationally efficient and avoids privacy concerns associated with payload inspection. GBDTs naturally handle categorical and numerical data, making them well-suited for diverse packet header attributes.

Hypotheses 2: Construction of the Data Exfiltration Model – Gradient Boosted Decision Trees in general can detect both Data Exfiltration and Keylogger attacks with an average F1-score of 0.95 compared to F1-score of 0.85 using Convolutional Neural Networks. GBDTs handle class imbalances better out of the box and can optimize weighted loss functions effectively. Neural Networks often need large datasets to learn patterns effectively, which may not always be available in cybersecurity use cases. Neural Networks require more hyperparameter tuning and longer training times compared to GBDTs, making them less appealing in quick-deployment scenarios.

Hypothesis 3: Best Algorithm – Gradient Boosted Decision Trees are the best algorithms for detecting Data Exfiltration attacks compared to Neural Networks. GBDTs (e.g., XGBoost, LightGBM, or CatBoost) are highly effective for structured/tabular data, which is typical in cybersecurity logs. They handle feature interactions and non-linear relationships well without much preprocessing. GBDTs are computationally efficient for

training and inference on smaller datasets compared to NNs. Data exfiltration datasets may not always be massive, making GBDTs a practical choice. GBDTs generally perform well with less data compared to Neural Networks, which usually require larger datasets to avoid overfitting and achieve high accuracy. However, Neural Networks can excel in unstructured data. If data exfiltration patterns are identified in unstructured formats (e.g., network traffic images, logs converted to sequences), Neural Networks (NNs), especially Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), can outperform GBDTs. Neural networks can handle very high-dimensional or sparse features better when sufficient training data is available. Neural networks excels in scenarios where manual feature engineering is challenging.

5.2 Conclusions

Packet header features (spkts, sbytes, dpkts, dbytes, dur, rate, pkts, bytes) are fundamental for network traffic analysis and can be effective in detecting Data Exfiltration attacks. However, the performance of the model will also depend on the specific type of dataset and the complexity of the attacks. Neural Networks can only detect Data Exfiltration attacks while GBDT is excellent in detecting Data Exfiltration. However, model performance can vary based on Dataset Quality and Size. Proper tuning of hyperparameters can significantly impact the model's accuracy. “GBDTs ability to handle complex relationships and its robustness to noise and outliers make it a strong choice for network intrusion detection. In conclusion, while GBDT has shown promising results in this specific scenario, it is crucial to experiment with different models and techniques to find the best approach for a given dataset and use case.” (Alsarhan, 2023). GBDT utilizes packet header features effectively for Data Exfiltration detection. We

show a clear comparison between GBDT and Neural Networks, using the F1-score as a robust metric for evaluating their performance on two distinct attack types (Data Exfiltration and Keylogger).

5.3 Contributions to the Body of Knowledge

There was an extremely limited number of records available for Data Exfiltration. This small number of record samples makes it “difficult for a machine learning models to learn patterns that can be applied to real world scenarios.” (Alsarhan, 2023). However, our praxis showed that Gradient Boosting Decision Trees (GBDTs) is capable of network traffic classification and data exfiltration detection even with extremely limited data samples. Neural Networks can be easily adapted to handle various types of network traffic data, including time-series and sequential data. “Neural Networks can automatically learn complex features from raw data, reducing the need for extensive feature engineering.” (Dolhopolov, 2024). With proper architecture and training, Neural Networks can achieve high accuracy in classifying network traffic. Gradient Boosting Machine is most effective in handling imbalanced datasets. While not as interpretable as decision trees, these models can still provide insights into feature importance. Gradient Boosting Machine is often faster to train and deploy compared to Neural Networks.

The following are the research contributions to the body of knowledge:

First, GBDTs can accurately identify both Data Exfiltration and Keylogger attacks with limited data samples. GBDTs are designed for structured/tabular datasets where features are explicitly defined and preprocessed. GBDTs do not require the massive data volumes often needed by deep learning models. They rely on boosting techniques to iteratively correct errors, which makes them robust with smaller datasets.

GBDTs can prioritize and optimize the use of a small number of critical features, making them efficient for anomaly detection in resource-constrained and data-scarce environments. GBDTs can quickly learn patterns from small datasets, making them practical for scenarios where attack samples are scarce. Therefore, GBDTs exhibit superior detection performance as compared to standard techniques such Neural Networks.

Second, GBDTs can efficiently monitor all network traffic by focusing on packet header features compared to deep packet analysis. Packet headers contain structured and limited data, such as source/destination IP addresses, ports, protocols, and flags, which are straightforward to extract and analyze. Unlike Deep Packet Analysis (DPA), GBDTs do not require the computationally expensive task of parsing and inspecting full packet payloads. By ignoring payload data, GBDTs respect privacy concerns, a critical factor in industries like healthcare and finance where inspecting packet contents might violate data protection regulations.

Third, Gradient Boosting Decision Trees can operate under compute resource constrained environments. GBDTs require fewer computations during inference compared to deep learning models, making them ideal for deployment on edge devices or systems with limited CPU/GPU power. Many implementations of GBDTs allow incremental or staged training, enabling updates without retraining the entire model. Unlike deep learning models, which often require GPUs or TPUs for efficient training and inference, GBDTs can operate efficiently on standard CPUs. Libraries like LightGBM and XGBoost are optimized for speed and memory usage. LightGBM, in

particular, is designed to handle large-scale data efficiently with low resource consumption.

5.4 Future Research

Developing large-scale, realistic datasets is crucial for training and evaluating models. (Sabir, et al. 2020). Lack of reliably labeled data makes it difficult to train accurate machine learning models for detecting and analyzing botnet activity. Poor attack diversity in the dataset inhibits realistic simulations of various network attacks, making it hard to develop comprehensive defenses. (Koroniotis, et al. 2019). Noisy or biased data may lead to poor performance in detecting data exfiltration using GBM. (Friedman, et al. 1999). gaNet-C can be useful for detecting data exfiltration which would need a labeled dataset created that includes examples of network traffic associated with data exfiltration activities and normal traffic. (Lopez-Martin, et al. 2023). An imbalanced dataset makes it difficult to train robust models as the algorithm might be biased towards classifying most instances as "normal." (Dong, et al. 2020). Lastly, challenges dealing with significant amounts of noise in the data can degrade performance. (Scheck, 2000).

MITRE CALDERA can simulate various stages of an attack, including initial compromise, lateral movement, data exfiltration, and post-exploitation activities. This enables datasets that mimic real-world scenarios, making it easier to train machine learning models for detecting novel exfiltration attacks. (Gjerstad, 2022). In contrast, MITRE ATT&CK follows a process-based model that provides standardized techniques and methodologies for finding threats across networks without requiring manual or ad-hoc scripting of processes on target systems. MITRE Caldera is lower level in terms of abstraction compared to MITRE ATT&CK as it requires security analysts to script out

their own processes, whereas MITRE ATT&CK provides a standardized set of processes and techniques that can be executed without requiring manual scripting. Both frameworks can integrate with various other threat hunting and information security tools, but MITRE Caldera may require more customization than MITRE ATT&CK to achieve the same level of integration or automation in some cases. MITRE Caldera primarily focuses on data collection aspects while MITRE ATT&CK provides a unified approach for threat hunting and analysis across networks using standardized techniques, processes, and methodologies. MITRE ATT&CK offers more customization options than MITRE Caldera since it allows security analysts to script their own processes without requiring manual scripting or ad-hoc scripting of existing ones. This makes MITRE ATT&CK a bit more flexible compared to MITRE Caldera, as security analysts can tailor the framework according to their specific needs and workflows. “The MITRE ATT&CK framework can enable execution automation of their process chain allowing for realistic simulation of network attacks to measure the effectiveness of implemented countermeasures and identify potential security gaps.” (Mundt, et al. 2022). Utilizing the MITRE ATT&CK framework can automate parts of their process chain, making it more efficient.

Neural Networks such as Autoencoders can generate a more realistic synthetic dataset which can improve network traffic classification that significantly improves Data Exfiltration detection performance. Autoencoders can learn compact representations of "normal" network traffic and generate realistic synthetic examples by decoding the learned latent space. By training on normal traffic, Autoencoders can reconstruct data patterns and simulate benign or slightly anomalous traffic for balanced training datasets.

This approach improves detection by enabling models to distinguish subtle anomalies like low-rate data exfiltration attempts.

General Adversarial Networks (GANs) consist of a generator and a discriminator network, where the generator produces synthetic samples, and the discriminator evaluates their authenticity. For network traffic, GANs can create highly realistic synthetic data that includes edge-case scenarios, such as sophisticated attacks. These synthetic datasets can enrich imbalanced datasets and improve model generalization, enhancing recall without inflating false positives.

Finally, a hybrid model using Gradient Boosting Decision Trees to focus on packet header features then applies Neural Networks to analyze packet payload complementing the strengths of both algorithms. GBDTs excel in analyzing structured data, such as packet header features (e.g., source/destination IPs, port numbers, protocol types). Packet header analysis often reveals patterns of abnormal traffic flow, making GBDTs an effective first-stage classifier. Neural Networks are adept at processing unstructured data, like packet payloads, where deeper patterns of anomalies can exist. Applying Neural Networks to packet payloads complements the header-focused analysis by identifying data exfiltration techniques like encrypted data transfer or obfuscated payloads. The hybrid model combines the strengths of both approaches. For example:

Stage 1: A GBDT detects anomalies using packet header features and forwards flagged packets for deeper payload inspection.

Stage 2: The Neural Networks identifies more complex patterns from GBDT anomalies and classifies the traffic using packet payload.

Advantages of the Hybrid Approach include Enhanced Detection. GBDTs quickly filter out normal traffic based on headers, reducing the computational load for Neural Networks . Neural Networks provide depth in detecting payload-based anomalies, handling cases where header features are insufficient. Neural Networks offer interpretable insights into feature importance (e.g., source/destination activity). The hybrid model allows better transparency in decisions compared to standalone Neural Networks. By delegating structured header analysis to GBDTs, the computational overhead of deep learning models is mitigated.

5.5 Future Applications of this Research

The banking and finance sectors are a critical component of a nation's economic infrastructure. Data exfiltration in this sector poses a significant threat to national security. Large-scale data breaches can erode public trust in financial institutions, leading to bank runs and economic instability. Exfiltrated financial data can be used to manipulate markets, causing significant economic damage. Theft of proprietary financial models, trading strategies, or customer data can erode a nation's competitive edge in the global financial market. Foreign intelligence agencies may target financial institutions to gain insights into a nation's economic vulnerabilities and strategic plans. Cyberattacks on financial institutions can be used as a tool of cyber warfare, disrupting economic activity, and undermining national security. Financial institutions often rely on critical infrastructure, such as payment systems and telecommunications networks. A breach in these systems could have cascading effects on national security. Gradient Boosting can be effectively applied to detect data exfiltration threats in the banking and finance sector. By sequentially adding weak predictive models, it can accurately identify anomalous

patterns in financial transactions and network traffic, indicating potential malicious activity. Gradient Boosting can be adapted to various types of financial data and attack scenarios. By effectively leveraging gradient boosting, financial institutions can significantly enhance their ability to detect and respond to data exfiltration attempts, safeguarding their sensitive information and protecting their customers.

Illegal diversion or manipulation of energy sources, including electricity, oil, and gas is a growing threat that has significant implications for national security, affecting not only the energy sector but also the economy, public health, and safety. Energy theft can lead to significant economic losses, undermining the stability of the energy market and potentially compromising the solvency of energy companies. Illegal connections and tampering with energy infrastructure can compromise grid reliability, increasing the risk of power outages, equipment damage, and even catastrophic failures. Energy theft can result in unsafe and unregulated energy diversion, posing risks to public health and safety, including electrocution, explosions, and fires. Energy theft often involves cyber-attacks, which can compromise the security of energy infrastructure, potentially leading to disruptions and sabotage. Energy theft can compromise the reliability and resilience of critical infrastructure, including hospitals, emergency services, and communication networks.

Energy theft detection is a critical task in the energy sector where Gradient Boosting can help identify suspicious consumption patterns. We can gather energy consumption data from smart meters or other sources, extract relevant features from the data, such as: Consumption patterns, Time of day, Day of the week, Seasonality, and

Weather data. We train a Gradient Boosting model using the engineered features then use the trained model to identify consumption patterns that deviate from the norm.

Communications networks are the backbone of modern society, connecting individuals, businesses, and governments. However, they are also vulnerable to cyberattacks, including data exfiltration, which poses a significant threat to national security. Adversaries can intercept sensitive communications, including government secrets, diplomatic cables, and military intelligence. Sensitive data, such as personal information, financial records, and intellectual property, can be stolen and analyzed by foreign intelligence agencies. Cyberattacks can disrupt telecommunications networks, leading to widespread communication failures and economic disruption. Malicious actors can compromise critical infrastructure components, such as power grids and transportation systems, which rely heavily on telecommunications. Hackers can spread misinformation and disinformation to influence public opinion and undermine democratic processes. Cyberattacks can target election infrastructure to manipulate election results or sow discord.

Trade secret theft poses a significant threat to national security, as it can compromise sensitive information and intellectual property (IP) vital to a nation's economic and technological advancement. This threat can have far-reaching consequences. Stolen trade secrets are often used by competitors to gain an unfair advantage, leading to lost sales and revenue. The theft of trade secrets can lead to the loss of jobs and damage to industries. The theft of trade secrets can discourage innovation and R&D investments. Stolen trade secrets can compromise sensitive IP, including patents,

copyrights, and trademarks. Trade secret theft can involve cyber-attacks, which can compromise national security.

Finally, Gradient Boosted Decision Trees can be used to detect data spills by identifying patterns and anomalies in data access logs. Collect data access logs from various sources, such as network logs, database logs, or application logs then extract relevant features from the logs, such as: User ID, Data accessed, Time of access, Location of access, Device used. Gradient Boosting can be used to identify patterns and anomalies in data access as well as detect unusual data access patterns that may indicate a data spill. Gradient Boosting can be used to identify unauthorized access attempts, such as brute-force attacks, SQL injection, and DDoS attacks. Identifying Data Exfiltration can help detect data exfiltration attempts, such as unauthorized file transfers or data leaks. Anomaly Detection can help identify unusual network traffic patterns that may indicate malicious activity. Insider Threat Detection can help detect malicious behavior from insiders, such as data theft or sabotage. By effectively leveraging gradient boosting, telecommunications providers can significantly enhance their ability to detect and respond to cyber threats, safeguarding their networks and protecting sensitive information.

References

Abhualganam. (2023). Real-time detection system for data exfiltration over DNS tunneling using machine learning. *University of Jordan*. Retrieved from https://www.mdpi.com/2079-9292/12/6/1467/pdf

Ahmed. (2019). Monitoring Enterprise DNS Queries for Detecting Data Exfiltration From Internal Hosts. *School of Electrical Engineering and Telecommunications, University of New South Wales*. Retrieved from https://www2.ee.unsw.edu.au/~hhabibi/pubs/jrnl/19TNSM_dnsExfil.pdf

Ahmed. (2019). Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks. *Electrical Engineering and Telecommunications, University of New South Wales*. Retrieved from https://www2.ee.unsw.edu.au/~hhabibi/pubs/conf/19im.pdf

Aldwairi. (2018). An evaluation of the performance of Restricted Boltzmann Machines as a model for anomaly network intrusion detection. *Computer Networks*. doi:https://doi.org/10.1016/j.comnet.2018.07.025

Alsarhan. (2023). Real-Time Machine Learning-based Intrusion Detection System (IDS) for Internet of Things (IoT) Networks. *The School of Engineering and Applied Science of The George Washington University*. Retrieved from https://scholarspace.library.gwu.edu/downloads/gx41mj73t?disposition=inline&locale=en

Al-shehari. (2024). Comparative evaluation of data imbalance addressing techniques for CNN-based insider threat detection. *Department of Self-Development Skill,*

Common First Year Deanship, King Saud University. doi:https://doi.org/10.1038/s41598-024-73510-9

Asian. (2023). Data Exfiltration Anomaly Detection on Enterprise Networks using Deep Packet Inspection. *Nusa Putra University*. Retrieved from https://journal.universitasbumigora.ac.id/index.php/matrik/article/view/3089/1787

Badirli. (2020). Gradient Boosted Neural Networks: GrowNet. *Department of Computer Science, Purdue University.* Retrieved from https://arxiv.org/pdf/2002.07971

Choudhary. (2018). Intrusion Detection Systems for Networked Unmanned Aerial Vehicles: A Survey. *Department of Information Security Engineering, Soonchunhyang University*. Retrieved from https://arxiv.org/pdf/1807.00435

Chukwu. (2023). Leveraging the MITRE ATT&CK Framework to Enhance. *Carleton University Ottawa*. Retrieved from https://repository.library.carleton.ca/downloads/rb68xd14n

Cody. (2022). Discovering Exfiltration Paths Using Reinforcement. *National Security Institute, Virginia Tech*. Retrieved from https://arxiv.org/pdf/2201.12416

Dolhopolov. (2024). Neural Network Threat Detection Systems for Data Breach Protection. *4th International Conference on Smart Information Systems and Technologies (SIST).* Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10629526

Dong. (2018). Gradient Boost Convolutional Autoencoder with Neural Decision Forest. *Journal of Latex Files.* Retrieved from https://arxiv.org/pdf/1806.08079

Dong. (2022). A neural network Boosted regression model based on XGBoost. *Applied Soft Computing*. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S1568494622003659

Emami. (2023). A Gradient Boosted Approach for Training Convolutional and Deep Neural Networks. *IEEE Open Journal of Signal Processing (Volume: 4)*. Retrieved from https://doi.org/10.1109/OJSP.2023.3279011

Feng. (2018). Multi-Layered Gradient Boosted Decision Trees. *National Key Lab for Novel Software Technology, Nanjing University*. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2018/file/Paper.pdf

Friedman. (2001). GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTED MACHINE. *Annals of Statistics, Standord University*. Retrieved from https://www.jstor.org/stable/2699986

Friedman. (2002). Stochastic Gradient Boosting Machine. *Computational Statistics and Data Analysis*. doi:https://doi.org/10.1016/S0167-9473(01)00065-2

Gjerstad. (2022). Generating labelled network datasets of APT with the MITRE CALDERA. *University of Oslo*. Retrieved from https://www.duo.uio.no/bitstream/handle/10852/95692/1/Gjerstad_thesis.pdf

Hinton. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science, 9, 147-169*. doi:https://doi.org/10.1016/S0364-0213(85)80012-4

Hinton. (1986). Learning representations by backpropagating errors. *Institute of Cognitive Science, University of California*. Retrieved from https://www.cs.toronto.edu/~hinton/absps/naturebp.pdf

Isakov. (2018). Preventing Neural Network Model Exfiltration in Machine Learning Hardware Accelerators. Retrieved from http://people.csail.mit.edu/mkinsy/papers/tie-asianhost18.pdf

Joback. (2020). A Statistical Approach to Detecting Low-Throughput Exfiltration through the Domain Name System Protocol. *Workshop in Dynamic and Novel Advances in Machine Learning and Intelligent Cyber Security*. doi:https://doi.org/10.1145/3477997.3478007

King. (2021). Data Exfiltration: Methods and Detection Countermeasures. *Cyber Security Research Group, University of Portsmouth*. Retrieved from https://pure.port.ac.uk/ws/portalfiles/portal/42722347/Data_Exfiltration.pdf

Koroniotis. (2019). Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. *Future Generation Computer Systems.* Retrieved from https://arxiv.org/pdf/1811.00701

Li. (2021). Approximate reasoning with fuzzy rule interpolation: background and recent advances. *Artificial Intelligence Review (2021) 54:4543–4590*. doi:https://doi.org/10.1007/s10462-021-10005-3

Lopez-Martin. (2019). Neural network architecture based on Gradient Boosting Machine for IoT traffic prediction. *Future Generation Computer Systems, Volume 100, 2019, Pages 656-673.* doi:10.1016/j.future.2019.05.060

Martinez. (2022). New data poison attacks on machine learning classifiers for mobile exfiltration. *Center for Cyber-Physical Systems, EECS Department, Khalifa University of Science and Technology*. Retrieved from https://arxiv.org/pdf/2210.11592

Martinez-Munoz. (2019). Sequential Training of Neural Networks with Gradient Boosting Machine. *Escuela Politectica Superior, Universidad Autonoma de Madrid.* Retrieved from https://www.researchgate.net/profile//links/Sequential-Training-of-Neural-Networks-with-Gradient-Boosted.pdf

McGinnity. (2020). AdaBoost-CNN: An adaptive Boosted algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. *Neurocomputing Volume 404, 3 September 2020, Pages 351-366.* doi:https://doi.org/10.1016/j.neucom.2020.03.064

Moghimi. (2016). Boosted Convolutional Neural Networks. *Cornell Tech, Cornell University.* Retrieved from http://www.svcl.ucsd.edu/publications/conference/2016/BMVC/Boosted_cnn.pdf

Molina. (2023). Machine Learning for Intrusion Detection of Cyber-Attacks in Unamanned Aerial Vehicles. *The George Wasginton University School of Engineering and Applied Sciences*. Retrieved from https://scholarspace.library.gwu.edu/downloads/vd66w090s?disposition=inline&locale=en

Mundt. (2022). Towards Mitigation of Data Exfiltration Techniques Using the MITRE ATT&CK Framework. *Esri Deutschland GmbH, Bonn*. Retrieved from https://www.unibw.de/digfor/publikationen/pdf/2021-12-icdf2c-mundt-baier.pdf

Nadler. (2018). Detection of malicious and low throughput data exfiltration over the DNS protocol. *Ben-Gurion University of the Negev*. Retrieved from https://arxiv.org/pdf/1709.08395

Natekin. (2013). Gradient Boosting Machine A Tutorial. *Frontiers in Neurorobotics.* Retrieved from https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/pdf

Qian. (2021). iPiDA-GBNN: Identification of Piwi-interacting RNA-disease associations based on Gradient Boosted neural network. *IEEE International Conference on Bioinformatics and Biomedicine.* Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9669592

Ramadhan. (2021). Comparative Analysis of ADASYN-SVM and SMOTE-SVM Methods. *Scientific Journal of Informatics*. Retrieved from https://pdfs.semanticscholar.org/ee1a/b5961d34882a130bd748d1d55c20290a2f1e.pdf

Rishu. (2023). Enhancing Exfiltration Path Analysis Using. *National Security Institute, Virginia Tech*. Retrieved from https://arxiv.org/pdf/2310.03667

Sarkar. (2022). XBNet: An extremely boosted neural network. *Intelligent Systems with Applications.* Retrieved from https://www.sciencedirect.com/science/article/pii/main.pdf

Schwenk. (2000). Boosted Neural Networks. *Neural Computation, LIMSI-CNRS, 91403 Orsay cedex.* Retrieved from https://www.researchgate.net/Boosted_Neural_Networks/links/Boosted-Neural-Networks.pdf

Tongsuwan. (2021). ConvXGB: A new deep learning model for classification problems based on CNN and XGBoost. *Nuclear Engineering Technology*. doi:https://doi.org/10.1016/j.net.2020.04.008

Willems. (2023). Data Exfiltration Detection on Network Metadata with Autoencoders. *National Cyber Security Centre (NCSC), Ministry of Justice and Security, 2511 DP The Hague*. Retrieved from https://www.mdpi.com/2079-9292/12/12/2584/pdf

Yu. (2023). Boosted Dynamic Neural Networks. *The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)*. Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/26302/26074

Zang. (2016). Scene Classification via a Gradient Boosted Random Convolutional Network Framework, vol. 54, no. 3. Retrieved from www.researchgate.net/profile/Fan-Zhang-233/publication/Scene-Classification-via-a-Gradient-Boosted-Random-Convolutional-Network-Framework.pdf

Zeeshan. (2022). Protocol-Based Deep Intrusion Detection for DoS and DDoS Attacks Using UNSW-NB15 and Bot-IoT Data-Sets. *Department of Computing, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad*. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9656911

Zhan. (2022). Detecting DNS over HTTPS based data exfiltration. *Institute of Information Engineering, Chinese Academy of Sciences*. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S1389128622001104

Appendix A

1.1 Multilayer Perceptron Neural Network

Python Code:

```
from keras.models import Sequential
from keras.layers import Dense

# Create a MLP model with two hidden layers
mlp = Sequential()
mlp.add(Dense(64, activation='relu', input_shape=(6,)))  # Input layer with 6 features ->
Hidden layer with 64 units and ReLU activation
mlp.add(Dense(32, activation='relu'))  # Hidden layer with 32 units and ReLU activation
mlp.add(Dense(2, activation='softmax'))  # Output layer with 2 units (normal/anomaly)
and softmax activation

# Compile the model
mlp.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

1.2 Deep Neural Network

Python Code:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

```
# Create a DNN model with three hidden layers and dropout regularization
dnn = Sequential()
dnn.add(Dense(128, activation='relu', input_shape=(6,)))  # Input layer with 6 features ->
Hidden layer with 128 units and ReLU activation
dnn.add(Dropout(0.2))  # Dropout regularization with 20% rate
dnn.add(Dense(64, activation='relu'))  # Hidden layer with 64 units and ReLU activation
dnn.add(Dropout(0.3))
dnn.add(Dense(32, activation='relu'))  # Hidden layer with 32 units and ReLU activation
dnn.add(Dropout(0.4))
dnn.add(Dense(2, activation='softmax'))  # Output layer with 2 units (normal/anomaly)
and softmax activation

# Compile the model
dnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

1.3 Convolutional Neural Network

Python Code:

```
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense

# Create a CNN model with two convolutional layers and max pooling
cnn = Sequential()
cnn.add(Conv1D(64, kernel_size=3, activation='relu', input_shape=(6, 1)))  # Input layer
with 6 features and 1 channel -> Convolutional layer with 64 filters and ReLU activation
```

```
cnn.add(MaxPooling1D(pool_size=2))  # Max pooling layer with pool size 2
cnn.add(Conv1D(32, kernel_size=3, activation='relu'))
cnn.add(MaxPooling1D(pool_size=2))
cnn.add(Flatten())  # Flatten the output of the convolutional and pooling layers.
cnn.add(Dense(128, activation='relu'))  # Hidden layer with 128 units and ReLU
activation
cnn.add(Dense(2, activation='softmax'))  # Output layer with 2 units (normal/anomaly)
and softmax activation

# Compile the model
cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```