

Detecting Application Layer Attacks on IEEE 802.11 Networks Using Machine Learning

2054584

Supervisor: Dr. Christo Panchev

WMG Cyber Security Centre

University of Warwick

May 2023

Abstract

This project aligns with the following CyBoK Skills: Network Security, Security Operations & Incident Management

Abbreviations

Aegan Wi-Fi Intrusion Dataset v3	AWID3
Artificial Intelligence	AI
Area Under Curve	AUC
Autoencoders	AE
Machine Learning	ML
Intrusion Detection System	IDS
Intrusion Prevention System	IPS
Neural Network	NN
Deep Neural Network	DNN
Multi-Layer Perceptron	MLP
K Nearest Neighbour	KNN
Random Forest	RF
eXtreme Gradient Boosting	XGBoost
Address Resolution Protocol	ARP
Domain Name Service	DNS
Transmission Control Protocol	TCP
User Datagram Protocol	UDP
Server Message Block	SMB
Secure Shell	SSH
Simple Service Discovery Protocol	SDDP
Stratified Cross Validation	S-CV
F-Score/F-Measure	F1
Man-in-the-middle	MITM
Denial Of Service	DoS

Contents

Abstract	ii
Abbreviations	iii
List of Figures	vii
List of Tables	vii
1 Introduction	1
1.0.1 Wireless Networks And Attacks	1
1.0.2 Intrusion Detection Systems	2
1.1 Research Questions and Objectives	2
2 Literature Review	4
2.1 Intrusion Detection Systems	4
2.2 Datasets	4
2.3 Detecting Network Attacks	5
2.4 Machine Learning Algorithms	6
2.4.1 Random Forest	7
2.4.2 K-Nearest Neighbor	7
2.4.3 XGBoost	7
2.4.4 Neural Networks	7
2.5 Summary	8
3 Methodology	9
3.1 AWID3 Dataset	9
3.2 Code Environment	9
3.3 Libraries	10
3.4 Feature Selection	11
3.4.1 Application Layer Features	11
3.4.2 802.11 Features	13
3.4.3 Non-802.11 Features	13
3.5 Dataset Manipulation	15
3.6 Data Pre-Processing	18
3.6.1 Encoding	18
3.6.2 Normalisation	18
3.7 Data Balancing	18
3.8 Data Split	19
3.9 Cross Validation	19
3.10 Evaluation Metrics	20

4	Experiments	23
4.1	Initial Modelling	23
4.2	Parameter Tuning	23
4.3	Classifiers	24
4.3.1	Random Forest (RF)	24
4.3.2	XGBoost	26
4.3.3	KNN Classifier	29
4.4	Neural Networks	30
4.4.1	Multi-Layer Perceptron (MLP)	30
5	Analysis Of Results	33
5.1	Comparison of Models	33
5.2	Model Interpretation	33
5.3	Limitations	33
5.4	Future Work	33
5.5	Random Forest	34
5.6	XGBoost	37
5.7	MLP	40
6	Conclusion	42
6.1	Summary Of Findings	42
6.2	Implications	42
6.3	Contributions	42
6.4	Limitations	42
6.5	Future Work	42
	Bibliography	43
	Appendices	46
A	Ethical Approval	46
B	Dataset Manipulation	47
B.1	CSV Combiner Script	47
B.2	Feature Extraction & Reduction	48
C	Conda Environments	50
C.1	Neural Networks - Apple Silicon	50
C.2	Classifiers	50
D	Data Preprocessing	51
D.1	MinMax Scaling	51
D.2	OHE Encoding	51
D.3	Label Encoding	52

D.4	Loading Dataset	52
E	Classifiers	54
E.1	K-Nearest Neighbor (KNN)	54
E.2	Random Forest	56
E.2.1	RF Model ID 0 - Raw Metrics	56
E.2.2	RF Model ID 1 - Raw Metrics	59
E.2.3	RF Model ID 1 - Raw Metrics	60
E.2.4	RF Model ID 2 - Raw Metrics	61
E.2.5	RF Model ID 3 - Raw Metrics	62
E.2.6	RF Model ID 4 - Raw Metrics	63
E.2.7	RF Model ID 5 - Raw Metrics	64
E.3	XGBoost	65
E.3.1	Stock 100% - XGBoost Raw Metrics	65
E.3.2	Stock 100% - XGBoost CM	66
E.3.3	Stock 100% - XGBoost Feature Importance	67
F	Neural Networks	68
F.1	MLP NN v1	68
F.1.1	MLP Neural Network	68

List of Figures

4.1	Training Time for KNN Classifier	29
-----	--	----

List of Tables

3.1	The selected set of application layer features.	12
3.2	The selected set of 802.11 features.	13
3.3	The selected set of Non-802.11 Features	14
3.4	Data Before Cleaning and Processing	16
3.5	Data After Cleaning and Processing	17
3.6	Data Model Split into Train and Test Sets	19
4.1	Parameters for Random Forest Classifier	25
4.2	RF Model Parameters	25
4.3	XGBoost Model Parameters	26
4.4	XGBoost GridSearch Parameters	27
4.5	MLP Model Parameters Part 1	31
4.6	MLP Model Parameters Pt2	32
5.1	RF S-CV Mean Metrics	34
5.2	RF Test Metrics	34
5.3	XGBoost S-CV Metrics	37
5.4	XGBoost Test Metrics	37
5.5	MLP S-CV Metrics	40
5.6	MLP Test Metrics	40

1 Introduction

The ongoing increase in IoT devices in homes and enterprise environments has seen a rise in the utilisation of wireless networks such as IEEE 802.11 networks, more commonly known as WiFi. As businesses and consumers seek to try out new devices and technologies, these manufacturers tend to focus more on improving performance and features and neglect security (Roundy 2021). As a result, this may weaken the security posture of an organisation or home to be more susceptible to attacks from malicious threat actors taking advantage of vulnerable devices in the network.

1.0.1 Wireless Networks And Attacks

The 802.11 standards have advanced and improved since their inception in 1997 in terms of security, however, despite this, Wi-Fi networks are still vulnerable to well-known attacks such as de-authentication attacks to disconnect all devices from a network, leading to more advanced attacks such as Man-in-the-middle attacks (MITM) or Denial Of Service (DoS) attacks. The introduction of Protected Management Frames (PMF) in 2009 (Electrical and Engineers 2009) helped to increase the security of management frames by using cryptography and integrity protection on de-authentication, disassociation and action management frames (Satam and Hariri 2021).

The introduction of WPA3 in 2018 (*WPA3 Specification Version 3.1* 2022), aimed to succeed WPA2, bringing new features and fixes to strengthen the security of wireless networks. More notably, Simultaneous Authentication of Equals (SAE) was introduced to provide a secure key negotiation and key exchange method based on the Dragonfly key exchange protocol in RFC 7664 (Harkins 2015), preventing dictionary or brute-forcing attacks as well as the (KRACK) Key-Reinstallation attack (Vanhoe and Piessens 2017) by providing perfect forward secrecy, ensuring that even if the private key is obtained, the data packets cannot be decrypted.

Research into WPA3 networks indicates that even features such as Protected Management Frames (PMF) and Simultaneous Authentication of Equals (SAE) methods of authentication have their shortcomings including being vulnerable to denial-of-service, side-channel, and downgrade attacks (Vanhoe and Ronen 2020).

1.0.2 Intrusion Detection Systems

Intrusion Detection Systems (IDS) are a common mechanism to defend against these attacks by analysing network traffic and determining if they are malicious or benign. There are typically two types of intrusion detection: signature-based and anomaly-based. Signature-based IDS monitors the network traffic for any suspicious patterns within data packets that match a known signature for an intrusion. This is usually via a database holding known intrusion attack patterns. Anomaly-based IDS creates an organisational benchmark of 'normal' as a baseline to help determine whether an activity is considered unusual or suspicious. This involves feeding the system with a large amount of data to learn an environment's regular usage patterns initially.

External tools such as Stratosphere IPS (SLIPS) developed by Garcia, Goma, and Babayeva (2015) at the Stratosphere Lab at CTU University of Prague seek to utilise a combination of behaviour patterns and machine learning such as Markov Chain models to detect malicious network traffic. Open-source implementations of wireless IDS such as Kismet (Kismet 2002) and OpenWIPS-ng (d'Otreppe 2011) also exist and serve a usage for both consumers and businesses.

Significant work and research have been seen recently into investigating and developing wireless intrusion detection systems using Machine Learning based algorithms utilising supervised, unsupervised and deep learning approaches in both wired and wireless networks. However, research on Intrusion Detection Systems utilising 802.11 and other network protocol features e.g. ARP, TCP & UDP, including application layer features such as HTTP, DNS, SMB etc lacks sufficient research.

This research seeks to investigate and evaluate different machine learning algorithms in detecting and classifying attacks launched at the application layer level on 802.11 wireless networks for a proposed intrusion detection system.

1.1 Research Questions and Objectives

The objectives for the project are as follows:

- To explore and analyse current literature and academic research utilising ML for intrusion detection systems for IEEE 802.11 networks.
- To examine and identify common machine learning algorithms used for the classification in the context of network attacks.

-
- To train a combination of supervised machine learning models to classify and detect a series of attacks launched from the application layer on 802.11 wireless networks.
 - To compare the performance of such models on the dataset, providing a recommendation for a proposed Wireless Intrusion Detection System (WIDS)

2 Literature Review

This section covers the existing research and reviews literature, papers and reports focusing on publicly available datasets, existing work and different machine learning algorithms. The literature reviewed details some of the methodologies and techniques used to develop existing models created for detecting network attacks on 802.11 wireless networks. The practical element of this dissertation is inspired by the following papers and literature.

2.1 Intrusion Detection Systems

Saskara et al. (2022) studies the performance of detecting 10 Denial Of Service attacks using Kismet on a Raspberry Pi using Aireplay-ng to generate a DoS attack on the target access point secured with WPA2/PSK, the experiment was repeated ten times. Using Kismet, the authors were able to successfully identify the attack with an average detection time of 3.42 seconds.

2.2 Datasets

Hnamte and Hussain (2021) discusses 37 public datasets and their suitability for building and training an IDS, limitations and restrictions. It was concluded that these datasets do not represent newer threats such as zero-day attacks. An optimal dataset should consist of well-labelled, up-to-date and public network traffic ranging from regular user activity to different attacks and payloads. It was proposed that using multiple datasets in different network environments and scenarios across a standard set of features could help to improve the accuracy of ML-based Network Intrusion Detection Systems.

The AWID3 data set (E. Chatzoglou, G. Kambourakis, and C. Kolias 2021) released in February 2021 seeks to build upon the existing AWID2 data set by evaluating various network attacks in an IEEE 802.11 enterprise network environment. These include higher-level layer attacks initiated from the link layer across multiple protocols and layers and newly discovered 802.11w attacks such as Krack, Kook, SSDP amplification, malware and even botnet attacks (Constantinos Kolias et al. 2016). Included with the dataset, are the Pairwise Master Key (PMK) and TLS Keys. Additionally, AWID3's concentration on enterprise networks includes the use of Protected Management Frames (PMF) that help to provide additional information during usage for an IDS.

Previous work and research into evaluating numerous machine learning algorithms have been conducted on the well-known older AWID2 data set (Constantinos Kolias et al. 2016), however with an overall lack of publicly available wireless network datasets, the introduction of AWID3 can help to bring new research and training data to help develop new machine learning models.

In the context of wireless networks, the AWID suite of datasets is widely recognised and used within academic research and literature, being one of the only extensive publicly available datasets on 802.11 enterprise networks with respect to application layer attacks, AWID3 is a strong candidate for investigating the development of an IDS using machine learning.

2.3 Detecting Network Attacks

Application Layer Attacks

Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. (2022) discusses the detection of application layer attacks using machine learning utilising the AWID3 dataset. The authors did not rely on optimisation or dimensionality-reducing techniques, only the six PCAPS containing application layer attacks were used and more specifically, no application layer features were used e.g. HTTP and DNS. These were classified and grouped under three main classes: Normal, Flooding and Other. This was justified due to these being usually encrypted and therefore not easily accessible, moreover, it raises concerns about privacy, requiring attention to ensure the data does not contain personally identifiable information or data unique to the environment. A research gap was identified as no previous work focused primarily on detecting the attacks originating from the application layer on the newer AWID3 dataset.

A set of 802.11 and non-802.11 features was evaluated using three classifiers (Decision Tree, Bagging and LightGBM) and two DNNs (Multi-Layer Perceptron (MLP) and Denoising stacked Autoencoders (AE)). Of the classifiers, Bagging produced the highest scoring AUC with the MLP DNN performing slightly better than the AE across the non-802.11 and 802.11 features. The feature importance was evaluated and irrelevant features were removed and tested in combination, resulting in better results across models.

5G Attacks

Mughaid et al. (2022) discusses the rise and need for protection of 5G based attacks, including rule-based methods and machine learning-based methods. However, these methods have limitations in terms of accuracy and efficiency. To address these issues, the paper "Improved dropping attacks detecting system in 5g networks using machine learning and deep learning approaches" proposes a new system that leverages machine learning and deep learning techniques to achieve a high detection accuracy. A 99% accuracy was achieved using KNN and 93% for DF and Neural Network.

Attack Classifications

Islam and Allayear (2022) utilised the AWID dataset to predict one of four attacks using the KNN classifier, the paper presented strong results for the "ARP" attack type, achieving the best accuracy with recall. The paper highlighted the importance of the pre-processing of data, feature selection, and choosing an appropriate classifier and oversampling method. The authors suggested that including additional features in the classification process and testing a more generalised model could improve a model's performance in future research and prevent the curse of dimensionality.

The work by Dalal et al. (2021) investigates WPA3 Enterprise Networks against a combination of known WPA3 attacks alongside a series of older WPA2 attacks such as Beacon Flood and De-authentication attacks. It was concluded that eight of the nine attacks to the testbed's Access Point were vulnerable and a chosen Intrusion Detection System was unable to identify and detect the attacks. Dalal et al. (ibid.) then proceeded to design a new signature-based IDS using Python. A packet capture of each attack was captured and processed into the proposed IDS, if there were indicators of attacks, the IDS outputted the time and classified the type of attack. The paper focuses on logical reasoning to deduce an attack rather than utilising anomaly detection such as machine learning.

2.4 Machine Learning Algorithms

A key area of the work was deciding the machine learning algorithms to use, a combination of classifiers and neural networks were considered in their context of suitability, efficiency and performance. AWID3 is a labelled dataset, as such only supervised algorithms were used for this work. The section below provides an overview of existing work utilising

the AWID3 dataset for the machine learning algorithms for detecting wireless network attacks and provides a justification for their usage in this project.

2.4.1 Random Forest

Random Forest is an ensemble learning algorithm that combines multiple decision trees during its training process, at each node the best features are selected to split the tree with additional pruning used to help prevent overfitting. The predictions of all the individual decision trees are combined to make a final prediction.

2.4.2 K-Nearest Neighbor

K-Nearest Neighbor is a non-parametric algorithm that works by finding the k closest neighbours to a given input and classifying it based on the majority class within the k neighbours. The works by Torres (2021) utilised KNN on the AWID2 & 3 datasets on a set of ten features. In order to save memory, only the last thousand samples were used. The model quickly converged at a high accuracy of 0.95 on AWID2 and 0.88 on AWID3.

2.4.3 XGBoost

XGBoost, short for eXtreme Gradient Boosting is a type of gradient-boosted decision tree. It was developed by Chen and Guestrin (2016) and is considered to be an efficient and scalable algorithm capable of handling large datasets and models. It utilises a collection, referred to as an ensemble, of decision trees combined to create a model capable of learning from the errors of the previous tree in a sequence.

2.4.4 Neural Networks

Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) works using a feed-forward artificial neural network that consists of an input layer, one or more hidden layers, and an output layer. Each layer within contains a given number of neurons that are connected together to additional layers through weighted connections.

2.5 Summary

Based on the literature review and research on the AWID3 dataset and wireless network attack classification, it appears that detecting application layer wireless network attacks using machine learning remains an under-researched area. In their previous work, Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. (2022) showed that combining 802.11 and non-802.11 features achieved high accuracy and AUC, without using application layer features such as DNS, SMB and HTTP etc. However, it remains to be investigated whether combining these application layer features can improve the accuracy of machine learning classifiers in identifying application layer attacks on 802.11 networks. Furthermore, the works fail to classify individually the method of attack, combining the six attacks under three classes: Normal, Flooding and Other. This project aims to address this research gap by exploring the feasibility of using application layer features to enhance the performance of machine learning classifiers for detecting application layer attacks on specifically the AWID3 dataset.

3 Methodology

3.1 AWID3 Dataset

- SSH Bruteforce - a brute-force attack was conducted against the radius server
- Botnet - The attack deployed a piece of malware and infects a number of STAs, once executed the victim's machine pings back to the C2 server.
- Malware - Malware was launched that aims to exploit a specific vulnerability in the target system to further escalate the attack, e.g., deploy ransomware.
- SQL Injection - The target is an external node and a malicious SQL query string is inputted into a web form of the target. The packet's HTTP POST and GET requests can reveal the SQL code query.
- SDDP Amplification - This attack consists of a DDoS attack using the Simple Service Discovery Protocol and uses Universal Plug and Play (UPnP) to trick all STAs of the wireless network to send a barrage of packets to each SDDP-enabled device. Every device then responds, eventually leading to a DoS.

3.2 Code Environment

The code for developing the machine learning models was programmed using Python 3.8/9, Visual Studio Code, and Jupyter Notebooks for the IDE. All experiments were conducted on a hardware combination of an M2 Mac Mini with 8 Cores and 16GB RAM or an Intel(R) Xeon(R) CPU E5-2699 VM running Ubuntu 22.04.02 LTS with 64 GB RAM and an Nvidia Tesla M40. Accordingly, the two machines will be referred to as 'M2' and 'VM'. Due to the limitations and errors encountered, TensorFlow GPU Acceleration was not utilised for Deep Learning on the M2 Mac Mini.

In order to create a reproducible environment and manage dependencies, Conda virtual environments (Distribution 2016) were used to isolate the experiments on the M2 Mac Mini. A TensorFlow GPU docker container running Nvidia CUDA was utilised on the VM. See Appx C for the full code for creating the environments.

3.3 Libraries

Several libraries were used to develop and implement the machine learning models, including: A selection of common machine learning libraries was utilised for this project, namely Numpy, Pandas, Scikit-Learn (Pedregosa et al. 2011), Matplotlib, Seaborn, Joblib, Jupyter, Tensorflow (Martín Abadi et al. 2015) and XGBoost (Chen and Guestrin 2016).

3.4 Feature Selection

Similar to the work carried out by Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. (2022), six attacks out of the 21 from AWID3 were concentrated, namely Botnet, Malware, SSH, SQL Injection, SSDP Amplification and Website Spoofing, these are attacks that originate from the application layer and forms a good scope of research for this project.

This work aims to combine the (16) 802.11 and (17) non-802.11 features from *ibid.* with a set of chosen application layer features with the aim to detect and classify the different application layer attacks. As previously established, existing research determined a high degree of accuracy and performance when combining both the 802.11 and non-802.11 features together, but a lack of research into determining if including additional application layer features would provide grounds for a further context into developing a machine learning model and affect its overall performance.

3.4.1 Application Layer Features

The AWID3 dataset contains 254 features within each of its attack CSV files, including application layer features in a decrypted format; provided by the decryption keys. While this may not be readily available in most cases, within an organization’s internal network in the context of an IDS, some application layer features will be accessible, such as any unencrypted DNS, HTTP, SMB, and NBNS traffic since the keys to protected 802.11 wireless networks would be available. However, to ensure data privacy and avoid bias from information specific to the AWID3 environment or containing identifiable information such as URLs and IP addresses, these features were not selected for this study. Therefore, the selected application layer features can be seen in Table 3.1. By combining these selected application layer features, this study aims to develop a machine learning classifier capable of accurately distinguishing between the different types of attacks.

Application Layer Features (19)		
Feature Name	Preprocessing Method	Data Type
nbns	OHE	object
ldap	OHE	object
dns	OHE	object
http.content_type	OHE	object
http.request.method	OHE	object
nbss.type	OHE	int64
smb2.cmd	OHE	int64
http.response.code	OHE	int64
ssh.message_code	OHE	int64
nbss.length	Min-Max	int64
dns.count.answers	Min-Max	int64
dns.count.queries	Min-Max	int64
dns.resp.len	Min-Max	int64
dns.resp.ttl	Min-Max	int64
ssh.packet.length	Min-Max	int64

Table 3.1: The selected set of application layer features.

The section below covers in more detail each of the selected features and their justification.

NetBIOS name service can be used to identify the names of machines on a network. The *nbns* feature combined with the *nbss.type* and *nbss.length* can provide context into the connections made between machines on a network without including AWID3 specific information. Different types of session packets can be indicative of certain activities such as file transfers and remote execution etc. The length of the packets can also help to identify any anomalous activity that may be useful for a machine learning classifier.

http.content.type, request.method and response.code: These features relate to the HTTP used for web browsing. They can provide insights into the type of content accessed by an attacker, the type of request method used, and the HTTP response code that was received. These HTTP features can be used to help identify potential attacks exploiting web-based vulnerabilities such as SQL Injections or Website Spoofing.

Domain Name System (DNS) is responsible for translating human-readable domain names to IP addresses. *dns.count.answers, count.queries, resp.len, and resp.ttl* chosen can provide additional information about DNS traffic, such as the number of queries and answers, the response length, and the time to live of each response. These can be used to help identify potential reconnaissance attacks and provide insights into the

network traffic patterns to identify potential DNS-based attacks such as DNS spoofing, cache poisoning, or tunnelling.

SMB (Server Message Block) is a client-server communication protocol used for sharing resources such as files and printers, in 2017 several Remote Code Execution vulnerabilities were discovered relating to the SMB protocol, including the wider known MS17-010 Eternal Blue exploit. By examining SMB activity, the *smb.cmd* we can determine different access types such as SMB access attempts, SMB file transfers, or SMB authentication requests, using this it may be possible to identify anomalous behaviour that could be indicative of an attack.

3.4.2 802.11 Features

The works by Efstratios Chatzoglou, Georgios Kambourakis, Constantinos Kolias, et al. 2022

802.11 Features (16)		
Feature Name	Preprocessing Method	Data Type
radiotap.present.tsft	OHE	int64
wlan.fc.ds	OHE	int64
wlan.fc.frag	OHE	int64
wlan.fc.moredata	OHE	int64
wlan.fc.protected	OHE	int64
wlan.fc.pwrmtg	OHE	int64
wlan.fc.type	OHE	int64
wlan.fc.retry	OHE	int64
wlan.fc.subtype	OHE	int64
wlan_radio.phy	OHE	int64
frame.len	Min-Max	int64
radiotap.dbm_antsignal	Min-Max	int64
radiotap.length	Min-Max	int64
wlan.duration	Min-Max	int64
wlan_radio.duration	Min-Max	int64
wlan_radio.signal_dbm	Min-Max	int64

Table 3.2: The selected set of 802.11 features.

3.4.3 Non-802.11 Features

Table 3.3 shows the non-802.11 features used in the analysis. It consists of Transport layer (TCP & UDP) protocols features responsible for data transfer and ARP features that operate on the Data-link layer to resolve Mac addresses. By analysing

Non-802.11 Features (17)		
Feature Name	Preprocessing	Data Type
arp	OHE	object
arp.hw.type	OHE	int64
arp.proto.type	OHE	int64
arp.hw.size	OHE	int64
arp.proto.size	OHE	int64
arp.opcode	OHE	int64
tcp.analysis	OHE	int64
tcp.analysis.retransmission	OHE	int64
tcp.checksum.status	OHE	int64
tcp.flags.syn	OHE	int64
tcp.flags.ack	OHE	int64
tcp.flags.fin	OHE	int64
tcp.flags.push	OHE	int64
tcp.flags.reset	OHE	int64
tcp.option_len	OHE	int64
ip.ttl	Min-Max	int64
udp.length	Min-Max	int64

Table 3.3: The selected set of Non-802.11 Features

3.5 Dataset Manipulation

The AWID3 Dataset (E. Chatzoglou, G. Kambourakis, and C. Kolias 2021) is supplied in two formats, a set of CSV files representing each method of attack and its subsequent data and the raw PCAP network captures. This project, as mentioned previously focuses on the six attack methods. For this instance, the CSV files were utilised and the dataset was manipulated to suit the purpose of experimentation. Each attack contained a folder with the data split into numerous CSV files, these needed to be rejoined to form one file/dataset so that it could be utilised and processed accordingly.

The methodology proposed was as followed:

1. Combine all individual CSV files for each attack method into one file using a bash script.
2. Import the file as a data frame and extract the desired features into a separate data frame.
3. Remove Nan and fix invalid values
4. Replace missing values to 0
5. Remove Nan target values.
6. Export the data frame as a new CSV file.
7. Combine all reduced datasets into one large data set.

Combing Files

A bash script, Appendix B.1 was created to list all contents of a given folder, containing the *.csv* file extension and sorted into numerical order i.e. 01, 02, 03. However, each individual file contained the CSV header, so only the first CSV file's header was read and written into the new '*combined.csv*' file. All other files were read and appended into the new file, ignoring the first line; the CSV header.

This step resulted in 6 large CSV files with the following rows and file sizes. See Table 3.4

Class	Rows	File Size
SSH	2,440,571	3 GB
Botnet	3,226,061	4.27GB
Malware	2,312,761	3.41GB
SQL Injection	2,598,357	3.8 GB
SSDP	8,141,645	8.02 GB
Website Spoofing	2,668,568	2.85 GB

Table 3.4: Data Before Cleaning and Processing

Feature Extraction

With the combined datasets, the selected features were extracted from the 254 features as referenced in Table 3.1, 3.2 and 3.3. Due to the large file sizes, numerous errors and kernel crashes were encountered during the importing of the file into Pandas.

Instead of importing all columns, the required features were specified using the *'use_cols'* parameter along with the *'chunk size'* parameter to read the file in smaller chunks to save memory and eventually combined them together, forming one dataframe. This saw a reduction in import time and lower memory consumption.

Data Cleaning

Following this, the data was cleaned to ensure it was fit for the next stage of data pre-processing. Rows that contained only NAN values were dropped, as well as missing Label values. All missing/nan values from each column were replaced and represented with 0, following a similar approach to Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. (2022).

Upon analysis, frequent occurrences of hyphenated values e.g *-100-100-10*, *123-456-1*, *-10-2*, *81-63-63* etc. were observed. These were more notable in the 802.11w features such as *'radiotap.dbm_antsignal'* and *'wlan_radio.signal_dbm'*, this was expected, as being wireless radio features, *'radiotap.dbm_antsignal'* represents the signal strength in decibel milliwatts (dBm) and is captured via multiple antennas each representing the captured signal strength. A similar approach to (ibid.) was followed, extracting and keeping the first value in the sequence, e.g *-100-100-10* became -100, *123-456-1* became 123, *-10-2* became -10 and *81-63-63* became 81. A regex expression was written to iterate through each column to replace these values accordingly.

Following on, invalid values were observed such as the presence of values containing months such as: *Oct-26*, *Oct-18*, *Feb-10* etc. This was determined to be a processing error during the creation of the

CSV files from the PCAP files and represented a low majority of the dataset. It was concluded that rows containing invalid values would be dropped from the data. A similar RegEX expression was written to filter out these values from the following columns: *'tcp.option.len'*, *'dns.resp.ttl'*, *'ip.ttl'*, *'smb2.cmd'*. The full code for this section can be found in Appendix B.2.

Individual Datasets

After data cleaning and processing, the final 6 individual data files consisted of the following. See Table 3.5

Class	Rows	File Size
SSH	2,433,851	298 MB
Botnet	3,216,505	393 MB
Malware	2,304,632	283 MB
SQL Injection	2,590,119	317 MB
SSDP	8,137,106	1.04 GB
Website Spoofing	2,666,406	340 MB

Table 3.5: Data After Cleaning and Processing

Combining Datasets

Finally, utilising the same bash script (B.1) the six reduced CSV files were combined into one large single data frame and then subsequently exported to a CSV file. The resulting file was 2.67GB in size and contained approximately 21,348,614 rows.

3.6 Data Pre-Processing

3.6.1 Encoding

One of the main decisions when building a model for a classification problem is the choice of encoding such as label, ordinal and one-hot encoding.

One-hot encoding was chosen to encode the categorical data for our models, a binary vector is created for each category, and at once only one element is set to 1 (referred to as 'Hot' i.e True) and the rest set to 0 (referred to as 'Cold' i.e. False). This approach will avoid assigning arbitrary numerical values to each variable that the model may interpret as having a weighting depending on its value.

Ensemble Classifiers such as Random Forest do not require the target variable i.e. Labels to be encoded and can be interpreted as a string e.g. Normal, SSH, Malware etc. However, for deep learning, K-Nearest Neighbor and XGBoost we also utilised One-Hot Encoding to encode the target variable. Refer to D.2 for the code used to One-Hot Encode the categorical features.

3.6.2 Normalisation

Scaling was performed on the dataset for normalisation to help normalise all numerical values and bring features to a similar scale. Min-Max scaler was chosen to scale the data between 0 and 1. As a linear scaling method, it helps preserve the original distribution's shape, ensuring it does not affect the underlying relationship between the different features in the data. Refer to D.1 for the code used to perform the MinMax scaler on the numerical features in the dataset.

3.7 Data Balancing

At its core, the dataset is imbalanced, with a majority of 'Normal' data with varying ranges of available malicious data from each attack class. Consideration was taken to utilise data balancing methods such as SMOTE and Random under/oversampling to help distribute the data. However, in a normal environment one would expect an overwhelming majority of Normal network traffic, therefore to best represent a real-life scenario, the data was kept imbalanced, ensuring changes were not made to the underlying distribution of the dataset. Refer to Table 3.6 for the distribution of each class respectively before and after splitting into the train and test sets.

3.8 Data Split

A stratified train-test split was performed on the dataset by splitting the entire dataset into training and testing sets to ensure the distribution of the target variable i.e. Label is the same in both sets. When training a machine learning model, the testing set is used to evaluate the performance of the model to help prevent overfitting. Overfitting can occur when the model learns all of the features and relationships of the training data; almost memorising the data. Subsequently, it struggles to predict new, unseen data.

Class	Train Data (70%)	Test Data (30%)	Whole Data (100%)
Normal	10,668,482	4,572,206	12,192,550
SDDP	3,849,896	1,649,955	4,399,881
Website Spoofing	283,576	121,533	324,087
Malware	92,112	39,476	105,270
Botnet	39,806	17,060	45,493
SSH	8,317	3,565	9,506
SQL Injection	1,840	789	2,103

Table 3.6: Data Model Split into Train and Test Sets

Analysing the split, we observe a large imbalance of data between each class of attack, in particular, SQL Injection makes up less than 0.01% of the entire dataset, with SSDP taking the majority 21% of the data.

3.9 Cross Validation

Due to the imbalanced nature of the datasets, stratified k-fold cross-validation with a k value of 10 was used, similar to the works carried out by Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. (2022). The training set is split into 10 folds, the model is then trained on all folds, except one called the validation set. The model is then tested on the validation set for its performance metrics and recorded. This is then repeated for all 10 folds, so each fold is used as a validation set. The results are then averaged to better represent the model's training performance across the data. Stratified split ensures each fold contains the same proportion of samples within each class to preserve the underlying structure of the data.

Finally, after Cross Validation, the model is trained using the full training set and evaluated based on the testing set to obtain a final measure of performance, before finally saving the model.

3.10 Evaluation Metrics

A key area of the work was deciding the use of specific metrics to evaluate the performance of the models. Metrics are vital to determine if models are under or over-fitting on the data and help to provide context into steps and modifications needed to improve the performances of the models. As a multi-class classification problem, the primary focus was on the two main metrics of evaluation AUC and F1. Given the nature of the problem, it is important to both minimise false negatives and false positives, which represents misclassifying the attack of the network traffic as either a potential intrusion (false negative) or falsely marking normal traffic as malicious (false positives). By looking placing a strong emphasis on the F1 and AUC scores, this aims to provide a balanced measure of the model's performance.

AUC-ROC

The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) measures the ability of a model to correctly distinguish between positive and negative classes. AUC-ROC is also insensitive to class imbalances. Similarly in the works carried out in (Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. 2022; Efstratios Chatzoglou, Georgios Kambourakis, Constantinos Kolias, et al. 2022) AUC was used as one of the primary evaluation metrics.

This value is first calculated by plotting the Receiver Operating Characteristic (ROC) curve using the True Positive Rate (TPR) against the False Positive Rate (FPR) for each classification threshold. The TPR is a measure of the proportions of positive values that were correctly classified. Similarly, the FPR is the proportion of negative values that are incorrectly classified as positive. Using the ROC curve, the area under the curve (AUC) is calculated. This value ranges between 0 and 1, where 0.5 represents at best random guessing, and 1 corresponds to a perfect classifier.

As our problem is multi-class, the AUC will be calculated by computing the one-vs-all metric for each class separately i.e., calculated for each class individually, treating all samples for that class as positive and all others as negative. Then these scores are averaged to calculate a final AUC score.

F1

The F1 score is a weighted average of both precision and recall. Precision is the fraction of correctly predicted positive instances out of all

total predicted positive instances. The recall is the fraction of correctly predicted positive instances out of the total actual positive instances. Tune the model to achieve a balance between false positives and false negatives.

Equations for Precision, Recall & F1

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

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

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Micro, Macro and Weighted

In regular binary classification, metrics such as F1, Precision, Recall and AUC can be calculated easily, however, for our multi-class classification problem, a slightly different approach must be taken. In particular, there are three main methods:

- Micro averaging uses the metric across all classes by counting the total true positives, false positives, and false negatives. This is the equivalent of using the accuracy i.e., fails to take into account class imbalances.
- Macro averaging calculates the metric in each class independently and then averages this for all classes, giving equal weight for all classes. It is used typically when all classes are equally as important, irrespective of the class size or any imbalances.
- Weighted averaging also calculates the metric for each class independently, but the average of the individual class scores is weighted with the number of samples in each class. It is used when performance across all classes is considered important, and the class imbalance needs to be considered.

Therefore, the weighted averaging method was chosen, leading to robust scores that take into account both the number of samples within the class and its performance. It was observed that most previous works fail to mention the averaging method used for its evaluation metrics.

Classification Report

In addition to viewing the averaged metrics across all classes, the classification report provides a comprehensive summary detailing the metrics for Precision, Recall, Accuracy and F1 across each class. This is important for understanding the underlying performance of the model, we can easily identify specific classes the model may be underperforming on, allowing for better tweaking and modifications to improve performance.

Confusion Matrix

The Confusion Matrix is a table that displays the performance of a model by showing the number of true positives, false positives, true negatives and false negatives for each class. In other words, how accurate the classifier is on each class and how it tends to wrong predict each class for another (confusion). By examining the confusion matrix, we can identify any specific classes that may require additional tuning or changes to the model to improve its performance. Works by Koço and Capponi (2013) introduced a new method using confusion matrices to measure and analyse the performance of cost-sensitive methods showing the importance of the confusion matrix in imbalanced data sets.

4 Experiments

In this study, a diverse set of machine-learning algorithms were utilised for this problem. Three shallow classifiers: Random Forest, K-Nearest Neighbour and XGBoost and one neural network: Multi-Layered Perceptron.

4.1 Initial Modelling

To speed up initial training and testing for each machine learning algorithm, a multitude of subsets of the original combined data were created using `sklearn`'s `train_test_split` to create a stratified split resulting in reduced datasets. Varying levels of data splits were created, including a 50%, 60% and 80% data split from the original 12 million rows of data as seen in Table 3.6. The reduced datasets allow for a quicker training time to determine the suitability of each model and helped to provide a rough measure of a model's underlying performance of the data. As discussed previously, where appropriate, additional Cross Validation is used during training to better understand the performance of the model.

4.2 Parameter Tuning

An important aspect of experimentation is the tuning of parameters specific to each model. Hyperparameters are defined during the creation of each model and have a substantial impact on its performance. As such, two primary methods are adopted. In some instances, an exhaustive searching method such as `GridSearchCV` is initially used to identify the optimal parameters. `GridSearchCV` uses a user-defined parameter grid and systematically evaluates each combination relative to the model's performance. `GridSearchCV` can be computationally expensive and time-consuming, due to limitations in both hardware, time and numerous crashes, `GridSearchCV` was not always used.

To address this issue, `RandomizedSearchCV` was adopted, by searching randomly through the parameter grid with a defined number of iterations, a good tradeoff is achieved that provides a balanced and efficient solution without sacrificing quality. Additionally, in some cases and initial experimentation, a combination of iterative experimentation and domain knowledge was used instead. This was justified due to limited time constraints and prior knowledge and understanding of the underlying algorithm.

4.3 Classifiers

4.3.1 Random Forest (RF)

In an attempt to find optimal parameters, GridSearchCV and RandomizedSearchCV were utilised. However, due to memory issues on both machines and frequent system crashes due to insufficient memory, experimentation for the Random Forest Classifier instead follows an exploratory approach, using both prior knowledge and iterative testing to create a series of models which were subsequently evaluated and optimised. The training was conducted on both the M2 and VM testbeds. Table 4.1 details the set of parameters focused on during experimentation, using performance feedback and the model’s behaviour on the dataset, these were tweaked and fine-tuned to help enhance the model’s performance. Table 4.2 documents the parameters used for all models.

Initial experimentation began using default parameters without changing or adding any values (Model 0-2). This helped to establish the baseline performance for subsequent comparison and analysis. This initial model achieved high S-CV average scores on the entire dataset. With such high metrics, it was important to ensure modifications made to the model avoided overfitting.

Subsequent experimentation was conducted by using the model’s training metrics along with the confusion matrices, classification reports and predictions on the test set. In Models 3 and 5, it was noted that the modification of the *class_weight* parameter from its default value to *‘weighted’* resulted in a decrease in S-CV performance across all metrics, moreover inspecting the metrics on the test set affirms this, with a higher number of misclassifications, but with higher recall and very low precision rates for several classes.

In the course of the iterative and exploratory phases, it was observed that additional models and future parameter tuning i.e. Models 3-5 all showed negative performance impacts when compared to the baseline model. This indicated that contrary to expectations, using additional parameter values did not increase the performance of this classifier. It was concluded that for the AWID3 dataset and this specific classification problem, the default parameters for the RandomForestClassifier showed superior performance. Subsequently, additional experimentation was concluded at this stage.

Table 4.1: Parameters for Random Forest Classifier

Parameter	Description
n_estimators	The number of trees in the forest
criterion	Function to measure the quality of a split.
max_depth	Maximum depth of the tree.
min_samples_split	Minimum samples required to split an internal node.
min_samples_leaf	Minimum samples required to be at a leaf node.
max_features	Maximum features to consider when splitting.
bootstrap	To bootstrap samples when constructing trees
class_weight	Weights associated with classes
random_state	The random seed.

Table 4.2: RF Model Parameters

Parameter	Model 0-2	Model 3	Model 4	Model 5
n_estimators:	100	100	200	100
max_depth:	None	10	15	10
min_samples_leaf:	1	2	1	2
min_samples_split:	2	3	2	3
random_state:	1234	1234	1234	1234
class_weight:	None	None	balanced	balanced

4.3.2 XGBoost

A series of models were trained and this model followed both an exploratory and iterative approach, RandomizedSearchCV was also utilised in the training phase to optimise hyperparameters. Table 4.3 shows the parameters used for each model, - denotes that no value was used, i.e., the default value was used.

Table 4.3: XGBoost Model Parameters

Parameter	Model 0-2&6	Model 3	Model 5	Model 8	Model 10	Model 11
early_stopping_rounds	-	10	10	10	10	10
subsample	-	-	-	-	0.9	-
n_estimators	-	-	300	300	200	300
min_child_weight	-	-	-	-	3	-
max_depth	-	-	5	5	9	5
learning_rate	-	-	0.2	0.2	0.3	0.2
gamma	-	-	-	-	0	-
colsample_bytree	-	-	-	-	0.7	-
reg_alpha	-	0.1	0.1	0.1	-	0.1

Initial experimentation began with the XGBoost classifier being trained with default parameters across a range of subsets of data, 60%, 80% and 100% as shown in models 0, 1 and 6. Additionally, model 1 further incorporated Stratified Cross Validation during training. This was used to establish clear baseline performance of the classifiers and helped to provide context when tuning parameters.

Where available, the VM machine was used for training, allowing XGBoost to maximise its performance on the dedicated GPU. Early stopping and regularisation were added in subsequent models to help avoid the model from overfitting on the training data. e.g. in Model 3 only early stopping and regularisation were added and no noticeable performance increase/decrease was observed.

Parameter Tuning

An attempt was made to utilise GridSearchCV for parameter optimisation; significant setbacks were encountered in achieving a successful execution due to numerous errors, system crashes, and exceptionally high grid search time. In light of the difficulties faced with GridSearchCV, RandomizedSearchCV was utilised.

Despite challenges faced with GridSearchCV, initial experimentation with 5 CV on a smaller parameter grid on the 80% dataset was

successful and subsequent parameters were tested with models 5 and 8. An additional RandomizedSearchCV was run on the entire dataset and combined with stratified 10-fold cross-validation to ensure the best-found parameters were verified and proved consistent across the 10 folds.

See Listing 1 for the parameter grids used.

```

1
2 gscv_param_grid = {
3     'learning_rate': [0.05, 0.1, 0.2],
4     'n_estimators': [100, 200, 300],
5     'max_depth': [3, 4, 5]
6 }
7
8 rgs_param_grid = {
9     'learning_rate': [0.01, 0.1, 0.3],
10    'max_depth': [3, 6, 9],
11    'min_child_weight': [1, 3, 5],
12    'gamma': [0, 0.1, 0.2],
13    'subsample': [0.5, 0.7, 0.9],
14    'colsample_bytree': [0.5, 0.7, 0.9],
15    'n_estimators': [100, 200]
16 }

```

Listing 1: Grid Search Parameters For XGBoost

Table 4.4: XGBoost GridSearch Parameters

Parameter	GS	RGS
Early Stopping	-	10
Evaluation Metric	merror	merror
Learning Rate	0.2	0.3
Max Depth	5	9
Min Child Weight	-	3
Gamma	-	0
Subsamples	-	0.9
Colsample By Tree	-	0.7
N Estimators	300	200

Best Found Parameters

The parameter tuning process helped to identify a set of optimal parameters using RandomisedSearchCV, as detailed in Table 4.4. These parameters were used to create Model 10. Subsequent models created afterwards, from additional parameter tuning did not show a sign of noticeable improvement in the model's performance. Due to limited

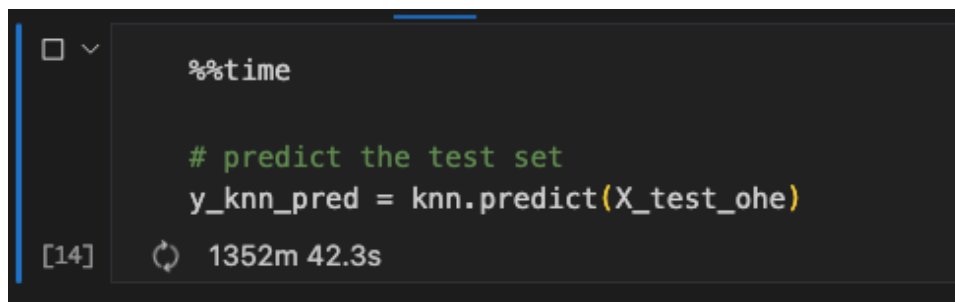
time and computational power, further parameter tuning was not performed, and the decision was made here to stop further experiments.

4.3.3 KNN Classifier

During initial experimentation, KNN took over 22 hours to predict on the test set following training - Figure 4.1. Additionally, utilising the VM machine to train, KNN took over 28 hours to predict the test set of data, and subsequently crashed the system multiple times before results or evidence could be gathered. KNN's algorithm means it does not build and store a model during training but rather stores them in memory. As a result, when predicting on the test set, it required high computational power as KNN searches for the K-nearest neighbour from the training set. Due to a large number of features, this further increases the computational power required for these tasks. This was deemed too long for real-world applications where detecting network attacks would be time-sensitive. As network attacks can occur quickly, an IDS using ML algorithms needs to have a quick response to detect these attacks.

Therefore, despite the advantages of KNN, such as being easy to implement and interpret, the prioritisation of speed and accuracy in this work led to the ultimate decision not to continue with this classifier. Consequently, the results for this classifier remain inconclusive.

Figure 4.1: Training Time for KNN Classifier

The image shows a Jupyter Notebook interface with a dark theme. On the left, there is a sidebar with a square icon and a dropdown arrow. The main area displays a code cell with the following content:

```
%%time  
  
# predict the test set  
y_knn_pred = knn.predict(X_test_oh)
```

 Below the code, the execution status is shown as `[14]` followed by a circular arrow icon and the time `1352m 42.3s`, indicating a very long execution time of over 22 hours.

4.4 Neural Networks

4.4.1 Multi-Layer Perceptron (MLP)

As part of the neural network experiments, Multilayered Perceptron models were created and tested through an exploratory process. It should be noted a wide range of MLP models was explored, the selection presented in the results section is a curated list, chosen for performance or notability. All models evaluated and their corresponding code can be found within the codebase. Table 4.5 and 4.6 details the set of parameters used for each notable MLP model.

Experimentation began with a 3 hidden layered MLP model consisting of 128, 64 and 32 neurons across the different subsets of data to gauge a rough estimate of the model's performance through varying levels of data. As such, cross-validation was not utilised. Models 0-3 consist of the same parameters tested across the 60 and 80% datasets, metrics were high, however, the models struggled to predict minority classes and resulted in low recall and F1-Scores. Performance when increasing the size of the dataset did improve the performance and can be attributed to the fact the larger dataset provided more samples of the minority class to be trained on. With this information, further models were created with increased batch sizes and epochs to train for longer.

Overfitting

A key aspect when training the MLP models was to prevent overfitting. To help mitigate this, techniques such as Early Stopping and Dropout were used in the majority of the models. Early Stopping was used during SCV, the training process monitors the validation AUC loss for signs of overfitting (e.g. when the model starts to learn the data and not generalising). Once the validation loss started to degrade over two defined epochs, the model would stop training. Dropout is a regularisation method that randomly sets 0.2 of the input neurons to 0. Dropout layers were used in the network architecture.

Thresholds

Towards the latter stages of experimentation, an attempt was made to further enhance the performance of the models on the misclassified classes. The individual class weightings were adjusted using the thresholds of each class. The aim was to identify the optimal threshold level between 0-1 that would maximise the F1 score for that class. A

systematic approach was followed to adjust the value in the class and evaluate the confusion matrices for changes in predictions.

Activator

Due to the nature of the problem (multi-class classification), applying existing knowledge and experience, the softmax activator was chosen for the output layer. It provides an easy-to-interpret output of the model as a list of probabilities for each class and uses the highest probability as the predicted class.

Tuning

The device used to train and the experiment was the M2 Mac Mini, experiments conducted on the VM were found to be slower and would frequently cause crashes, even when utilising the dedicated GPU. As such, the hardware and time constraints restricted the level of tuning and parameter searching that could be performed. Techniques such as GridSearchCV and RandomisedSearchCV were not feasible when combined with 10 Fold S-CV.

Due to the complexity and computational demands of running machine learning models, practical limitations such as time constraints result in fewer tested models than desired. After conducting a vast amount of experiments and achieving high-performance results, the decision was made to conclude further model experimentation.

Table 4.5: MLP Model Parameters Part 1

Parameter	Model 0-3	Model 4	Model 5
Asctivator:	ReLU	ReLU	ReLU
Output Activator:	Softmax	Softmax	Softmax
Initialiser:	he_uniform	he_uniform	he_uniform
Optimiser:	Adam	Adam	SGD
Momentum:	N/A	N/A	N/A
Early Stopping:	N/A	2	2
Dropout:	0.2	0.2	0.2
Learning Rate:	0.001	0.001	0.01
Loss:	CC	CC	CC
Batch Norm:	True	True	True
Hidden Layers:	3	3	4
Nodes per Layer:	128/64/32	128/64/32	256/128/64/32
Batch Size:	180	200	132
Epochs:	15	20	20

Table 4.6: MLP Model Parameters Pt2

Parameter	Model 6	Model 7
Activator:	LeakyReLU	ReLU
Output Activator:	Softmax	Softmax
Initialiser:	he_uniform	-
Optimiser:	Adam	SGD
Momentum:	N/A	0.9
Early Stopping:	2	2
Dropout:	0.2	0.25*3/0.2*2
Learning Rate:	0.01	0.01
Loss:	CC	CC
Batch Norm:	True	True
Hidden Layers:	4	5
Nodes per Layer:	256/128/64/32	100/80/60/40/20
Batch Size:	132	170
Epochs:	20	20

5 Analysis Of Results

In this section, the performance of the machine learning models on the test set is analysed and discussed. The models are evaluated using the metrics discussed previously, such as F1-Score, Area Under Curve (AUC), Precision, Recall and Accuracy. The best-performing models and algorithms are identified and interpreted. Finally, limitations and challenges faced in the analysis are discussed and addressed and areas of improvement for future work are suggested.

5.1 Comparison of Models

5.2 Model Interpretation

5.3 Limitations

5.4 Future Work

5.5 Random Forest

Six notable models were trained during experimentation with the Random Forest Classifier with a series of parameters and values. Tables 5.1 and 5.2 show the S-CV and Test metrics for AUC, F1, Precision, Recall and Accuracy. Each model's individual metrics, classification report, confusion matrix and feature importances can be found in Appendix E.2.

Table 5.1: RF S-CV Mean Metrics

Model ID	Size	AUC	F1	Precision	Recall	Accuracy
1	100%	99.99	99.66	99.66	99.68	99.67
3	100%	99.99	99.66	99.66	99.67	99.67
4	100%	99.95	95.23	98.50	92.96	92.96
5	100%	99.87	91.53	98.42	86.65	86.65

Table 5.2: RF Test Metrics

Model ID	Size	AUC	F1	Precision	Recall	Accuracy
0	80%	99.99	99.66	99.66	99.67	99.67
1	100%	99.99	99.66	99.66	99.67	99.67
2	100%	99.99	99.66	99.66	99.67	99.67
3	100%	99.99	99.66	99.66	99.67	99.67
4	100%	99.95	98.48	92.54	94.97	92.54
5	100%	99.86	91.17	98.48	86.01	86.01

Models 0-2

Models 0-2 all used the default parameters for the RandomForestClassifier, and therefore share similar results across metrics.

Despite model 0 being trained on an 80% subset of the data, its performance was similar to models 1 and 2, achieving test metrics of AUC: 99.99%, F1: 99.66%, Precision: 99.66%, Recall: 99.67% and Accuracy of 99.67%. The model achieved a perfect score for SDDP, with only seven misclassifications. Examining the classification report shows low recall for less represented classes such as Botnet and SSH, with recalls of 0.77 and 0.78, this is further verified by the confusion matrix. Models 1 and 2 were trained on the entire dataset, with the exception that model 1 was trained with 10 Fold Stratified Cross Validation meanwhile model 2 was not. Interestingly despite the change in dataset sizes, the models appear to perform nearly identically with the same consistently high metrics across both Cross Validation and Testing, suggesting the models are not overfitting. As seen in model 0, the

classification reports and confusion matrix shares similar performances, with model 1 and 2 having a smaller number of misclassifications i.e. from 7 false positives to 2 on the SDDP class. This may indicate that despite adding more data to training, the RandomForestClassifier was unable to learn from the extra information which leads to diminishing returns for this specific problem.

Class Weight

During experimentation, models 3 and 5 share almost identical parameters except for the parameter: *class_weight*. The Class Weight parameter allows for the classifier to handle imbalanced datasets, its default value is *None*, meaning the model treats every class as equal during the training process. Alternatively, when set to *'balanced'*, the model assigns high weights that are inversely proportional to the class frequencies (Pedregosa et al. 2011). Model 5's class weight is set to *balanced* compared to *None*. When evaluating the results, model 3 supersedes model 5 across almost every metric, with a higher F1, precision, recall and accuracy score with the following percentage decrease per metric: AUC: 0.12%, F1: 8.18%, Precision: 1.24%, Recall: 12.99%, Accuracy: 12.99%. In terms of classification, Model 3 struggled with some of the minority classes such as Botnet, SSH and Malware, on the other hand, model 5 had a higher recall for these classes, but suffered at the cost of a reduced precision for the Normal class. This suggests that even though higher weighting is given to the majority class, it does not necessarily lead to a better model in this scenario.

It was proposed this was due to Random Forest's majority voting decision factor, so although minority classes may have had a higher weighting, the class that has fundamentally more samples will have more trees *'voting'* for that class.

Feature Importance

For all the models, the feature importance of each model was collected and inferred. The feature importance provides a score for each metric and highlights how important each feature was to the creation of the random forest models. In particular, the top five features were as follows:

- *ip.ttl* Appeared in the top five for all models and was the number one feature for models 1,2,3 and 5. The TTL value in the dataset may be containing a series of patterns relating to the different network attacks.
- *http.request.method_M-SEARCH* also appeared as one of the top features across a few of the models and was the number one feature in model 0.

-
- *udp.length* This feature appeared in the top three features except for model 5.
 - *radiotap.dbm_antsignal* was in the top five features for all models, gaining number one importance in model 5.
 - *wlan_radio.duration* was also prevalent in the top five features across most models.
 - *frame.len* was also prevalent in the top five features across most models.
 - *wlan_radio.signal_dbm* and *wlan_radio.duration* gained an increase in performance across model 5, this could be the effects of adjusting the class weighting to *balanced* for the model.

Overall the Random Forest models showed strong performances across the classes, however, due to the imbalanced nature of the dataset, it may have hidden the weaknesses within the minority classes (e.g. SQL Injection & SSH). Iterative and exploratory experimentation showed that the default parameters achieved superior results compared to the other models. Future work should focus on using GridSearchCV and RandomisedSearchCV to provide more advanced parameter tuning. Moreover, further emphasis can be placed on the minority classes to help lower the number of false positives.

5.6 XGBoost

Table 5.3 summarises the average metrics with 10 Fold Stratified Cross-Validation and Table 5.4 shows the metrics across the 30% test set. Due to the numerous models created during experimentation, only the most notable models are included in the tables. The raw metrics for all models can be found in E.3.

Table 5.3: XGBoost S-CV Metrics

Model ID	Dataset	AUC	F1	Precision	Recall	Accuracy
6	100%	99.99	99.64	99.64	99.64	99.64
8	100%	99.99	99.64	99.64	99.65	99.65
10	100%	100.00	99.65	99.65	99.66	99.66
11	100%	100.00	99.64	99.64	99.65	99.65

Table 5.4: XGBoost Test Metrics

Model ID	Dataset	AUC	F1	Precision	Recall	Accuracy
0	60%	99.99	99.63	99.64	99.64	99.64
1	80%	99.99	99.64	99.65	99.65	99.65
2	80%	99.99	99.64	99.64	99.64	99.64
3	80%	99.99	99.64	99.64	99.64	99.64
5	80%	99.99	99.64	99.65	99.65	99.65
6	100%	99.99	99.65	99.65	99.65	99.65
8	100%	99.99	99.65	99.65	99.65	99.65
10	100%	99.99	99.65	99.65	99.66	99.66
11	100%	99.99	99.65	99.65	99.65	99.65

Models 0, 1 and 6 were trained across varying levels of data sizes, but the models shared similar performance metrics. Model 0, trained on 60% of the dataset slightly underperformed in classifying minority classes as seen in the F1, precision and recall scores from the classification report. Model 1 showed a similar pattern, but had a minor increase in correct classifications, especially for SSH. Model 6, being trained on more data, subsequently achieved a better result across most classes and can be seen in the classification report and confusion matrix. Precision and recall in the testing set were also increased. Although minor, the gradual improvement shows the positive impacts more data can have to help a model train and learn the patterns and complexities of the dataset, helping it to generalise well on unseen data.

In Model 3, 80% of the dataset was used and early stopping of 10 rounds and regularisation of 0.1 were added to the model. However,

analysing the metrics in detail models 1/2 share a similar performance. Both equally have high precision and recall and the weighted averages are very similar, the confusion matrices show some minor differences but were not significant to affect performance. It appears that the addition of early stopping and regularisation to model 3 did not significantly affect the performance of the model compared to the baseline.

GridSearchCV

As mentioned previously, one instance of GridSearchCV ran successfully that tested a combination of the learning rate, number of estimators and the max depth. The test was cross-validated 5 times and took 28.27 hours to complete. Models 5 (80%) and 8 (100%) were created with these parameters. Compared to their baseline counterparts (4 and 6), the results are incomparable. Model 5 shares identical values for AUC and F1, with a 0.1 increase in Precision, Recall and Accuracy, this is also similar in Model 8, except AUC rounds to 100. To summarise, the models with default parameters (Models 4 and 6) and the models with the best-found parameters from GridSearchCV (Models 5 and 8) have similar performances across both datasets. Model 11 adopts the same parameters with the added inclusion of S-CV, early stopping and regularisation, however, there were no significant improvements with similar highly metrics and behaviour.

RandomisedSearchCV

A parameter grid was tested with RandomisedSearchCV and took a total of 41.81 hours to complete. Each parameter combination was subjected to 10-fold Stratified Cross Validation to ensure its performance was measured fairly.

A new model was created with the best-found parameters, Model 10. The model performed very well, with an average AUC of 99.99 on the training data and 99.98 on the test data. The F1 score was 99.65 on the test set indicating the model has a high proportion of correct predictions, balancing both precision and recall well. The confusion matrix verifies this and shows the model to perform well for most classes, especially Normal, SDDP and Web Spoofing with almost perfect precision and recall. However, there are a few misclassifications for 'Botnet', 'Malware' and 'SSH'. The model struggled and occasionally misclassified Normal traffic as malicious, but performed well in the SQL Injection class, especially given the small number of samples. The Cross-validation and test set results are similar and indicate the model

is not overfitting and generalising well to new data. Using the total number of instances of each class, the misclassification report can be calculated for each and is shown accordingly:

- Botnet: $4208 / 17060 = 25\%$
- Malware: $7161 / 39476 = 18\%$
- Normal: $6365 / 4572206 = 0.0013\%$
- SQL: $89 / 789 = 11\%$
- SSDP: $0 / 1649955 = 0\%$
- SSH: $771 / 3565 = 21\%$
- WebSpooof: $3046 / 121533 = 2.5\%$

Feature Importance

In conclusion, the XGBoost classifier demonstrated a high level of performance across the classes for this classification problem. Despite the levels of imbalance, the models still held up relatively well. Techniques such as GridSearchCV and RandomisedSearchCV were used to fine-tune parameters; however, it became evident that despite this, the baseline parameter model could still deliver remarkably similar results. Future investigations may consider a bigger search grid with more focused tuning based on the specific characteristics of the classes or tasks.

5.7 MLP

In exploring Neural Networks, a series of MLP models were created with a varying number of parameters and was tested with different subsets of the dataset. Each model consisted of a different number of hidden layers and neurons, optimisers (Adam & SGD), regularisation techniques (Dropout and Early Stopping), learning rates etc. The primary metrics for evaluating the models were AUC and F1, but the classification and confusion matrices were considered and used to form a detailed picture of each model's performance on the individual attack classes. Table 5.5 and 5.6 show the S-CV and Test Set results for 8 MLP NN models.

Table 5.5: MLP S-CV Metrics

Model ID	Dataset	AUC	F1	Precision	Recall	Accuracy
4	100%	99.90	99.37	99.40	99.42	99.42
5	100%	98.40	94.68	96.85	95.49	95.49
6	100%	99.79	99.27	99.31	99.33	99.33
7	100%	99.72	99.23	99.25	99.31	99.31

Table 5.6: MLP Test Metrics

Model ID	Dataset	AUC	F1	Precision	Recall	Accuracy
0	60%	99.99	99.36	99.38	99.41	99.41
1	60%	99.99	99.34	99.36	99.38	99.38
2	80%	99.86	99.42	99.44	99.39	99.44
3	80%	99.98	99.39	99.44	99.44	99.44
4	100%	99.94	99.42	99.44	99.46	99.46
5	100%	99.88	99.36	99.37	99.40	99.40
6	100%	99.80	99.28	99.40	99.42	99.42
7	100%	99.84	99.29	99.33	99.35	99.35

Data Subsets

Examining the results across the different data subsets, both 60% and 80% models showed high precision and recall. Class-specific performances for minority classes were consistently low. Notable, in model 3 on the 80% subset there was an increase in recall within the SQL Injection class. The models exhibited high overall performance but struggled with frequent misclassifications which suggest more data is required for the model to correctly identify the specific classes.

LeakyReLU

Models 5 and 6 differ in the selection of the activation function. (ReLU in model 5 and LeakyReLU in model 6). Upon initial examination, there are differences in test metrics, but larger differences appear when looking at class-specific performances. Whilst the precision was increased in some classes such as Botnet and SSH, recall suffered substantially such as 0.13 for SQL Injection, indicating the model was able to reduce some false positives at the high cost of failing to identify the true positives.

Previous Works

The works by Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. (2022) similarly used an MLP model consisting of four hidden layers, these specifications were adopted in Model 7 to provide context. The model displayed an AUC of 99.84, F1 of 99.28, Recall of 99.35 and Accuracy of 99.35 on the test set. However, the precision, recall and F1 for SQL Injection are substantially lower at 0.02 and 0.05 compared to other classes. The model fails to identify this class accurately, similarly, Botnet and Malware also saw a drop in performance. The Confusion matrix further affirms this observation with a large number of predictions from those classes being misclassified as Normal traffic. This further emphasises the importance of tuning parameters and settings that are specific to the problem at hand.

Summary

The Multi-Layer Perceptron in TensorFlow can provide a vast array of parameters and options, leading to an endless number of combinations to be tailored and optimised. Finding the 'best' model in the classification problem is a difficult non-trivial task.

With challenges and limitations in the hardware and time, the approach for these experiments consisted of creating a sequence of models and comparing the performance metrics to previous models and the overall domain knowledge and task.

Experimentation stopped after a vast number of models were tested and reached diminishing returns. Whilst this approach does not guarantee the 'best' MLP configuration, it provides a practical and effective method that strikes a balance between complexity and constraints. Further work can be investigated into utilising parameter searching such as GridSearchCV to automate the process.

6 Conclusion

6.1 Summary Of Findings

Summarize the main findings of your research, including the performance of your machine learning models, insights gained from the analysis, and any limitations or areas for future work.

6.2 Implications

Discuss the implications of your research for the field of machine learning and for the specific problem you addressed. This could include discussing how your research advances the state-of-the-art, how it could be applied in practice, or how it could inform future research.

6.3 Contributions

Discuss the contributions of your research to the field of machine learning. This could include discussing how your research fills a gap in the literature, how it provides new insights into the problem you addressed, or how it advances the methodology used in the field.

6.4 Limitations

Discuss the limitations of your research and potential sources of error. Explain how these limitations could impact the generalizability of your findings and suggest ways to address these limitations in future research.

6.5 Future Work

Discuss potential future work that could be done to build on your research and improve the performance of machine learning models in the specific problem you addressed and in the field more broadly.

Bibliography

- Chatzoglou, E., G. Kambourakis, and C. Kolias (2021). “Empirical evaluation of attacks against IEEE 802.11 enterprise networks: The AWID3 dataset”. In: *IEEE Access* 9, pp. 34188–34205. DOI: 10.1109/ACCESS.2021.3061609.
- Chatzoglou, Efstratios, Georgios Kambourakis, Constantinos Kolias, et al. (2022). “Pick Quality Over Quantity: Expert Feature Selection and Data Preprocessing for 802.11 Intrusion Detection Systems”. In: *IEEE Access* 10, pp. 64761–64784. DOI: 10.1109/ACCESS.2022.3183597.
- Chatzoglou, Efstratios, Georgios Kambourakis, Christos Smiliotopoulos, et al. (2022). “Best of Both Worlds: Detecting Application Layer Attacks through 802.11 and Non-802.11 Features”. In: *Sensors* 22:15. ISSN: 1424-8220. DOI: 10.3390/s22155633. URL: <https://www.mdpi.com/1424-8220/22/15/5633>.
- Chen, Tianqi and Carlos Guestrin (2016). “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: <http://doi.acm.org/10.1145/2939672.2939785>.
- d’Otreppe, Thomas (Nov. 2011). *OpenWIPS-ng*. OpenWIPS-ng. URL: <https://openwips-ng.org> (visited on 10/01/2022).
- Dalal, Neil et al. (2021). “A Wireless Intrusion Detection System for 802.11 WPA3 Networks”. In: *CoRR* abs/2110.04259. arXiv: 2110.04259. URL: <https://arxiv.org/abs/2110.04259>.
- Distribution, Anaconda Software (2016). *Anacoda*. Anaconda Software Distribution. URL: <https://anaconda.com> (visited on 12/15/2022).
- Electrical, Institute of and Electronics Engineers (2009). “IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 4: Protected Management Frames”. In: *IEEE Std 802.11w-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, and IEEE Std 802.11y-2008)*, pp. 1–111. DOI: 10.1109/IEEESTD.2009.5278657.

-
- Garcia, Sebastian, Alya Gomaa, and Kamila Babayeva (Dec. 2015). *Slips, behavioral machine learning-based Python IPS*. Stratosphere IPS. URL: <https://www.stratosphereips.org/stratosphere-ips-suite> (visited on 09/29/2022).
- Harkins, Dan (Nov. 2015). *Dragonfly Key Exchange*. Tech. rep. 7664. 18 pp. DOI: 10.17487/RFC7664. URL: <https://www.rfc-editor.org/info/rfc7664> (visited on 09/30/2022).
- Hnamte, Vanlalruata and Jamal Hussain (Nov. 2021). “An Extensive Survey on Intrusion Detection Systems: Datasets and Challenges for Modern Scenario”. In: *2021 3rd International Conference on Electrical, Control and Instrumentation Engineering (ICECIE)*, pp. 1–10. DOI: 10.1109/ICECIE52348.2021.9664737.
- Islam, Tariqul and Shaikh Muhammad Allayear (2022). “Capable of Classifying the Tuples with Wireless Attacks Detection Using Machine Learning”. In: *Intelligent Computing Systems*. Ed. by Carlos Brito-Loeza et al. Cham: Springer International Publishing, pp. 1–16. ISBN: 978-3-030-98457-1.
- Kismet (July 2002). *Kismet*. URL: <https://www.kismetwireless.net> (visited on 09/30/2022).
- Koço, Sokol and Cécile Capponi (13-15 Nov 2013). “On multi-class classification through the minimization of the confusion matrix norm”. In: *Proceedings of the 5th Asian Conference on Machine Learning*. Ed. by Cheng Soon Ong and Tu Bao Ho. Vol. 29. Proceedings of Machine Learning Research. Australian National University, Canberra, Australia: PMLR, pp. 277–292. URL: <https://proceedings.mlr.press/v29/Koco13.html>.
- Kolias, Constantinos et al. (2016). “Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset”. In: *IEEE Communications Surveys & Tutorials* 18.1, pp. 184–208.
- Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: <https://www.tensorflow.org/>.
- Mughaid, Ala et al. (Sept. 2022). “Improved dropping attacks detecting system in 5g networks using machine learning and deep learning approaches”. In: *Multimedia Tools and Applications*. ISSN: 1573-7721. DOI:

-
- 10.1007/s11042-022-13914-9. URL: <https://doi.org/10.1007/s11042-022-13914-9>.
- Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.
- Roundy, Jacob (May 2021). *IoT Security: IoT Device Security Challenges & Solutions*. Verizon Enterprise. URL: <https://enterprise.verizon.com/resources/articles/iot-device-security-challenges-and-solutions> (visited on 11/21/2022).
- Saskara, Gede Arna Jude et al. (Feb. 2022). “Performance of Kismet Wireless Intrusion Detection System on Raspberry Pi”. In: EAI. DOI: 10.4108/eai.27-11-2021.2315535.
- Satam, Pratik and Salim Hariri (2021). “WIDS: An Anomaly Based Intrusion Detection System for Wi-Fi (IEEE 802.11) Protocol”. In: *IEEE Transactions on Network and Service Management* 18.1, pp. 1077–1091. DOI: 10.1109/TNSM.2020.3036138.
- Torres, Anibal (2021). *WiFi Anomaly Behavior Analysis Based Intrusion Detection Using Online Learning*. URL: <http://hdl.handle.net/10150/666297>.
- Vanhoef, Mathy and Frank Piessens (2017). “Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. Dallas, Texas, USA: Association for Computing Machinery, pp. 1313–1328. ISBN: 9781450349468. DOI: 10.1145/3133956.3134027. URL: <https://doi.org/10.1145/3133956.3134027>.
- Vanhoef, Mathy and Eyal Ronen (2020). “Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd”. In: *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 517–533. DOI: 10.1109/SP40000.2020.00031.
- WPA3 Specification Version 3.1* (Nov. 2022). Tech. rep. Wi-Fi Alliance. URL: <https://www.wi-fi.org/file/wpa3-specification> (visited on 10/03/2022).

A Ethical Approval

1 Dear 2054584,

2

3 Warwick ID Number: 2054584

4

5 This is to confirm that your Supervisor's Delegated Approval
form has been received by the WMG Full-time Student Office,
Detecting IEEE 802.11 Attacks using Machine Learning does NOT
require ethical approval.

6

7 You are reminded that you must now adhere to the answers and
detail given in the completed WMG SDA ethical approval form (
and associated documentation) within your research project.
If anything changes in your research such that any of your
answers change, then you must contact us to check if you need
to reapply for or update your ethical approval before you
proceed.

8

9 If your data collection strategy, including the detail of any
interview/ survey questions that you drafted changes
substantially prior to or during data collection, then you
must reapply for ethical approval before your changes are
implemented.

10

11 When you submit your project please write N/A against the
ethical approval field in the submission pro-forma and
include a copy of this email in the appendices of your
project.

12

13 Kind regards

14

15 Jade Barrett

B Dataset Manipulation

B.1 CSV Combiner Script

```
1 #!/bin/bash
2
3 # Input Directory
4 input_dir=" ../ Malware"
5
6 cd "$input_dir"
7
8 # Set the output file name
9 output_file="combined.csv"
10
11 # Check if the output file already exists and delete it
12 if [ -f "$output_file" ]; then
13     rm "$output_file"
14 fi
15
16 # Print a status message
17 echo "Combining files ..."
18
19 # Loop through all the files that match the pattern reduced_*.
20     csv
21 for file in $(ls *.csv | sort -V)
22 do
23     # Check if the file exists
24     if [ -f "$file" ]; then
25         # Print a status message
26         echo "Combining $file ..."
27
28     # If this is the first file , copy the header to the output file
29     if [ ! -f "$output_file" ]; then
30         head -n 1 "$file" > "$output_file"
31     fi
32
33     # Append all the rows except the header to the output file
34     tail -n +2 "$file" >> "$output_file"
35 fi
36 done
37
38 # Print a status message
39 echo "Done."
```

B.2 Feature Extraction & Reduction

```
1 # Define the columns to extract
2 cols_to_use = [
3     'frame.len', 'radiotap.dbm_antsignal', 'radiotap.length',
4     'wlan.duration', 'wlan_radio.duration', 'wlan_radio.signal_dbm',
5     'radiotap.present.tsft', 'wlan.fc.type', 'wlan.fc.subtype',
6     'wlan.fc.ds', 'wlan.fc.frag', 'wlan.fc.moredata',
7     'wlan.fc.protected', 'wlan.fc.pwrmtgt', 'wlan.fc.retry',
8     'wlan_radio.phy', 'udp.length', 'ip.ttl',
9     'arp', 'arp.proto.type', 'arp.hw.size',
10    'arp.proto.size', 'arp.hw.type', 'arp.opcode',
11    'tcp.analysis', 'tcp.analysis.retransmission', 'tcp.option.len',
12    'tcp.checksum.status', 'tcp.flags.ack', 'tcp.flags.fin',
13    'tcp.flags.push', 'tcp.flags.reset', 'tcp.flags.syn',
14    'dns', 'dns.count.queries', 'dns.count.answers',
15    'dns.resp.len', 'dns.resp.ttl', 'http.request.method',
16    'http.response.code', 'http.content_type', 'ssh.message_code',
17    'ssh.packet_length', 'nbns', 'nbss.length',
18    'nbss.type', 'ldap', 'smb2.cmd',
19    'smb.flags.response', 'smb.access.generic_read',
20    'smb.access.generic_write', 'smb.access.generic_execute',
21    'Label']
22
23 # Define the chunk size you want to read in each iteration
24 batch_size = 1000000
25
26 # Initialize an empty dataframe to hold the combined results
27 combined_df = pd.DataFrame()
28
29 # Iterate through the file in batches
30 for chunk in pd.read_csv('botnet_combined.csv', chunksize=
    batch_size, usecols=cols_to_use, low_memory=False):
31
32     # Combine the processed chunk with previous chunks
33     combined_df = pd.concat([combined_df, chunk])
```

```
1 # Drop all missing rows that contain only nan values
2 combined_df = combined_df.dropna(how='all')
3
4 # Drop all rows with missing values in Label Column
5 combined_df = combined_df.dropna(subset=['Label'])
6
7 # Fill NAs with zeros
8 # Change nan values to 0
9 combined_df = combined_df.fillna(0)
```

```

1 # Duplicate the dataframe
2 df = combined_df.copy()
3
4 # Regex to keep only the first value e.g
5 # -100-100-10 becomes -100, 123-456-1 becomes 123, -10-2
   becomes -10, 81-63-63 becomes 81
6 def seperated_values(x):
7     x = str(x)
8     match = re.match(r'^(-?\d+).*$', x)
9     if match:
10         return match.group(1)
11     else:
12         return x
13
14 # Go through all columns and change seperate values into just
   one value
15 for column in df.columns:
16     df[column] = df[column].apply(seperated_values)
17     print('Processing', column)
18 print('Done')
19
20 # Find Rows that contain values such as Oct-26, Oct-18, Feb-10
   etc.. as these appear to be invalid and we will drop these
   rows.
21 regex = r"\b(?:\d{2}|(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|
   Nov|Dec))-(?:\d{2}|(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct
   |Nov|Dec))\b"
22
23 # Use str.match method to apply the regex pattern to the column
24 mask = df['tcp.option.len'].astype(str).str.match(regex).fillna(
   False)
25 df = df[~mask]
26
27 mask = df['dns.resp.ttl'].astype(str).str.match(regex).fillna(
   False)
28 df = df[~mask]
29
30 mask = df['ip.ttl'].astype(str).str.match(regex).fillna(False)
31 df = df[~mask]
32
33 mask = df['smb2.cmd'].astype(str).str.match(regex).fillna(False)
34 df = df[~mask]
35
36 df.to_csv('Botnet_Reduced.csv', index=False)

```

C Conda Environments

C.1 Neural Networks - Apple Silicon

```
1 conda create -n nn-env python=3.9
2 conda activate nn-env
3 conda install -c apple tensorflow-deps
4 conda install -c conda-forge -y pandas jupyter
5 pip install tensorflow-macos==2.10
6 pip install numpy, matplotlib, scikit-learn, scipy, seaborn
```

C.2 Classifiers

```
1 # Conda environment used for Random Forest, XGBoost and K-NN.
2
3 conda create -n ml-env python=3.9
4 conda activate ml-env
5 conda install -c conda-forge -y pandas jupyter
6 pip install numpy, matplotlib, scikit-learn, scipy, seaborn,
   xgboost
```

D Data Preprocessing

D.1 MinMax Scaling

```
1 # Define the scaler
2 scaler = MinMaxScaler()
3
4 # Fit the scaler to the following columns we define
5 scale_cols = [
6     'frame.len',
7     'radiotap.dbm_antsignal',
8     'radiotap.length',
9     'wlan.duration',
10    'wlan-radio.duration',
11    'wlan-radio.signal-dbm',
12    'ip.ttl',
13    'udp.length',
14    'nbss.length',
15    'dns.count.answers',
16    'dns.count.queries',
17    'dns.resp.ttl',
18    'ssh.packet.length']
19
20 # Fit the X_train and X_test
21 X_train[scale_cols] = scaler.fit_transform(X_train[scale_cols])
22 X_test[scale_cols] = scaler.transform(X_test[scale_cols])
```

D.2 OHE Encoding

```
1 cols_to_encode = [col for col in X_train.columns if col not in
2     scale_cols]
3
4 X_all = pd.concat([X_train, X_test], axis=0)
5
6 X_all_ohe = pd.get_dummies(X_all, columns=cols_to_encode,
7     drop_first=True, dtype=np.uint8)
8
9 # split back into train and test sets
10 X_train_ohe = X_all_ohe[:len(X_train)]
11 X_test_ohe = X_all_ohe[len(X_train):]
```

D.3 Label Encoding

```
1 # Use Label Encoder to encode the target variable
2 le = LabelEncoder()
3
4 label_encoder = le.fit(y_train)
5 y_train_encoded = label_encoder.transform(y_train)
```

D.4 Loading Dataset

```
1 chunk_size = 1000000
2 dtype_opt = {
3     'frame.len': 'int64',
4     'radiotap.dbm_antsignal': 'int64',
5     'radiotap.length': 'int64',
6     'radiotap.present.tsft': 'int64',
7     'wlan.duration': 'int64',
8     'wlan.fc.ds': 'int64',
9     'wlan.fc.frag': 'int64',
10    'wlan.fc.moredata': 'int64',
11    'wlan.fc.protected': 'int64',
12    'wlan.fc.pwrmtg': 'int64',
13    'wlan.fc.type': 'int64',
14    'wlan.fc.retry': 'int64',
15    'wlan.fc.subtype': 'int64',
16    'wlan_radio.duration': 'int64',
17    'wlan_radio.signal_dbm': 'int64',
18    'wlan_radio.phy': 'int64',
19    'arp': 'object',
20    'arp.hw.type': 'object',
21    'arp.proto.type': 'int64',
22    'arp.hw.size': 'int64',
23    'arp.proto.size': 'int64',
24    'arp.opcode': 'int64',
25    'ip.ttl': 'int64',
26    'tcp.analysis': 'int64',
27    'tcp.analysis.retransmission': 'int64',
28    'tcp.checksum.status': 'int64',
29    'tcp.flags.syn': 'int64',
30    'tcp.flags.ack': 'int64',
31    'tcp.flags.fin': 'int64',
32    'tcp.flags.push': 'int64',
33    'tcp.flags.reset': 'int64',
34    'tcp.option_len': 'int64',
35    'udp.length': 'int64',
36    'nbns': 'object',
37    'nbss.length': 'int64',
38    'ldap': 'object',
39    'smb2.cmd': 'int64',
```

```

40     'dns': 'object',
41     'dns.count.answers': 'int64',
42     'dns.count.queries': 'int64',
43     'dns.resp.ttl': 'int64',
44     'http.content_type': 'object',
45     'http.request.method': 'object',
46     'http.response.code': 'int64',
47     'ssh.message_code': 'int64',
48     'ssh.packet_length': 'int64'
49 }
50
51 # Read the data
52 print('Reading X...')
53 X = pd.DataFrame()
54 for chunk in pd.read_csv('X.csv', chunksize=chunk_size, usecols=
    dtype_opt.keys(), dtype=dtype_opt, low_memory=False):
55     X = pd.concat([X, chunk])
56
57 print('Reading y...')
58 y = pd.DataFrame()
59 for chunk in pd.read_csv('y.csv', chunksize=chunk_size, usecols
    =['Label'], dtype='object', low_memory=False):
60     y = pd.concat([y, chunk])
61
62 # Split the data into training and testing sets
63 print('Splitting the data...')
64 X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.30, random_state=1234, stratify=y)

```

E Classifiers

E.1 K-Nearest Neighbor (KNN)

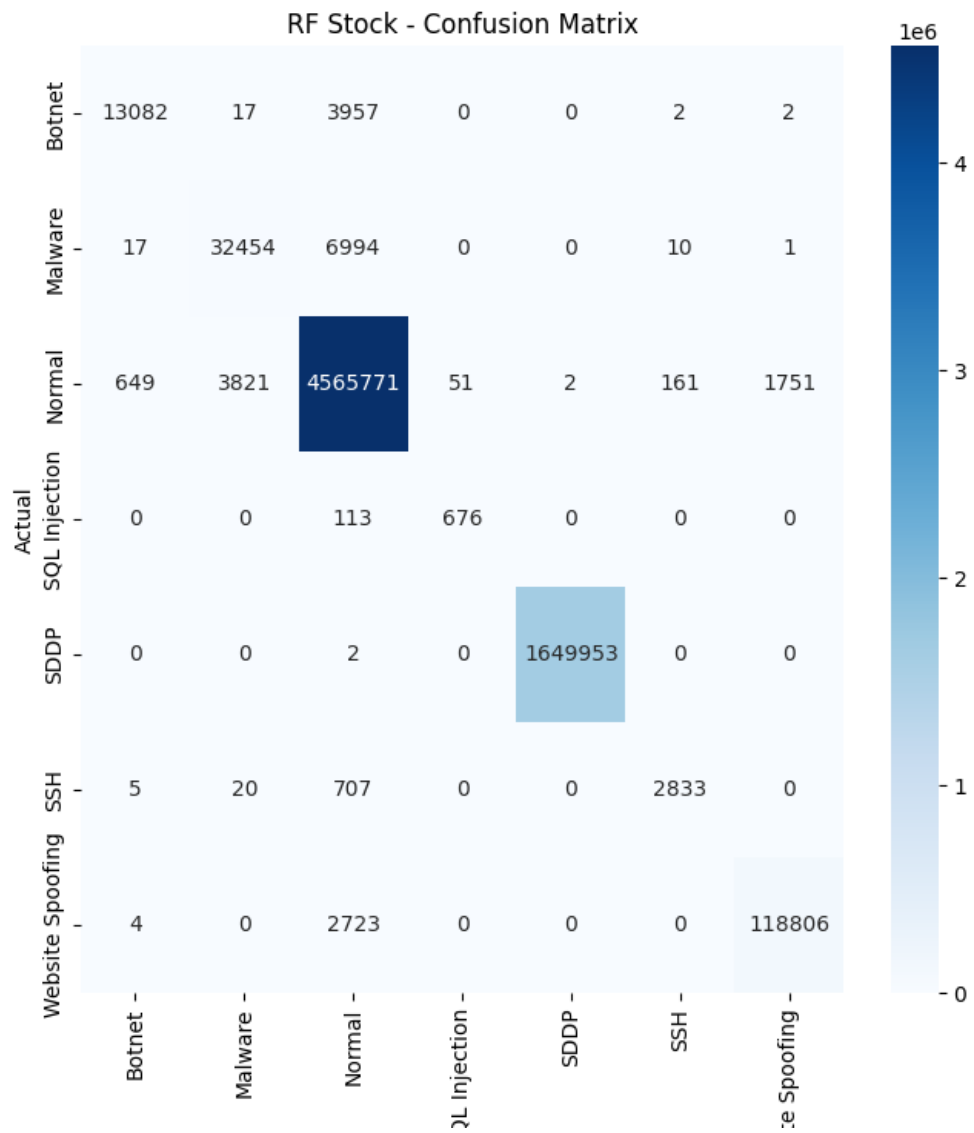
```
1 # Use KNN
2 from sklearn.neighbors import KNeighborsClassifier
3
4 k=5
5
6 # Create KNN classifier
7 knn = KNeighborsClassifier(n_neighbors=k, n_jobs=-1)
8
9 # Fit the model
10 knn.fit(X_train_ohe, y_train_encoded)
11
12 # predict the test set
13 y_knn_pred = knn.predict(X_test_ohe)
14
15 from sklearn.metrics import classification_report, roc_auc_score
16
17 # Get the classification report
18 report = classification_report(y_test_encoded, y_knn_pred)
19
20 print('Classification Report:\n', report)
21
22 # Get the all the metrics for the multi class classification
23
24 print('Accuracy: ', accuracy_score(y_test_encoded, y_knn_pred))
25 print('Precision: ', precision_score(y_test_encoded, y_knn_pred,
26                                     average='macro'))
26 print('Recall: ', recall_score(y_test_encoded, y_knn_pred,
27                                average='macro'))
27 print('F1 Score: ', f1_score(y_test_encoded, y_knn_pred, average
28                               = 'macro'))
28
29 # Get the confusion matrix for multi-class and plot it
30 confusion = confusion_matrix(y_test, y_rf_pred)
31 print('Confusion Matrix\n')
32 print(confusion)
33
34 # Plot the confusion matrix for multi-class classification using
35     seaborn
36
37 labels = ['Normal', 'SSDP', 'Website Spoofing', 'Malware', '
38     Botnet', 'SSH', 'SQL Injection']
39
40 plt.figure(figsize=(8, 8))
41 sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',
42             xticklabels=labels, yticklabels=labels)
43 plt.title('Confusion Matrix')
44 plt.xlabel('Predicted')
45 plt.ylabel('Actual')
46 plt.show()
47
```

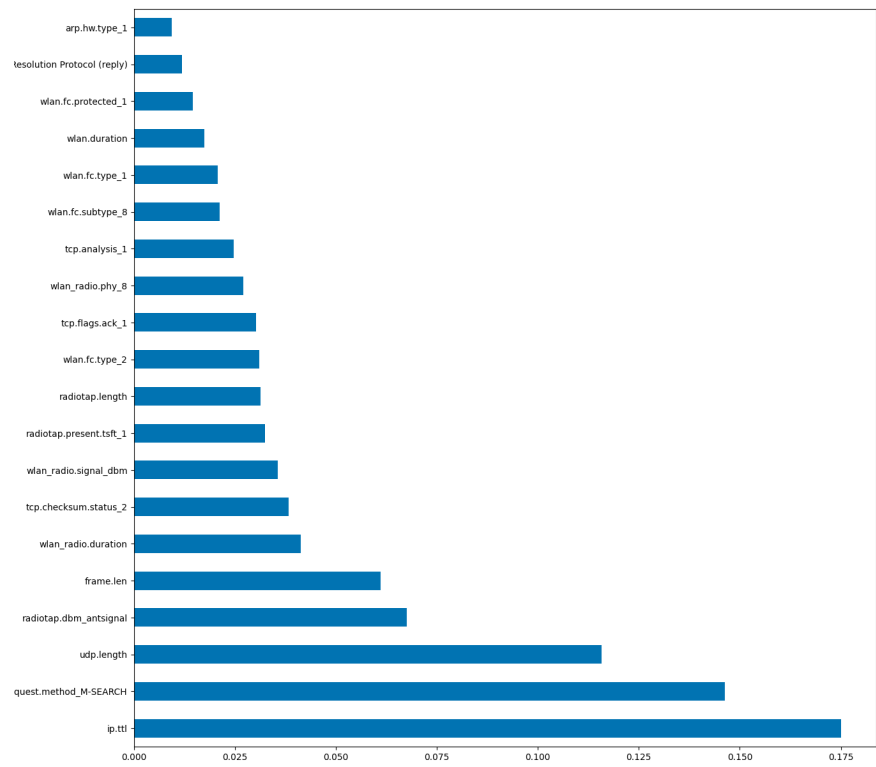
```
44 plt.figure(figsize=(10, 10))
45 feat_importances = pd.Series(rf.feature_importances_, index=
    X_train_ohe.columns)
46 feat_importances.nlargest(20).plot(kind='barh')
47 plt.show()
```

E.2 Random Forest

E.2.1 RF Model ID 0 - Raw Metrics

```
1 S-CV Results
2 Mean AUC = 99.99
3 Mean F1 = 99.66
4 Mean Precision = 99.66
5 Mean Recall = 99.67
6 Mean Accuracy = 99.67
7 Training Time: 7795 seconds
8
9 Final Test Results
10 Weighted AUC: 0.9999070506312879
11 Weighted F1: 0.996638797834701
12 Weighted Precision: 0.9966379719195173
13 Weighted Recall: 0.9967196932696956
14 Accuracy: 0.9967196932696956
15
16 Classification Report
17               precision    recall  f1-score   support
18
19      Botnet           0.95       0.77       0.85       17060
20      Malware           0.89       0.82       0.86       39476
21      Normal           1.00       1.00       1.00      457220
22      SQL              0.93       0.86       0.89        789
23      SDDP             1.00       1.00       1.00     1649955
24      SSH              0.94       0.79       0.86        3565
25      Spoofing         0.99       0.98       0.98     121533
26
27      accuracy                   1.00     6404584
28      macro avg           0.96       0.89       0.92     6404584
29      weighted avg        1.00       1.00       1.00     6404584
30
31 Confusion Matrix
32 [[ 13082    17   3957    0    0    2    2]
33 [    17  32454   6994    0    0   10    1]
34 [   649   3821 4565771   51    2   161  1751]
35 [    0    0   113   676    0    0    0]
36 [    0    0    2    0 1649953    0    0]
37 [    5   20   707    0    0  2833    0]
38 [    4    0  2723    0    0    0 118806]]
```





E.2.2 RF Model ID 1 - Raw Metrics

```
1 S-CV Results
2 Mean AUC = 99.99
3 Mean F1 = 99.66
4 Mean Precision = 99.66
5 Mean Recall = 99.67
6 Mean Accuracy = 99.67
7 Training Time 7794.549654006958 seconds
8
9 Final Test Results
10 Test AUC: 0.9999070506312879
11 Weighted Test F1: 0.996638797834701
12 Weighted Test Precision: 0.9966379719195173
13 Weighted Test Recall: 0.9967196932696956
14 Test Accuracy: 0.9967196932696956
15
16 Classification Report
17      precision    recall  f1-score   support
18
19     Botnet         0.95      0.77      0.85      17060
20     Malware         0.89      0.82      0.86      39476
21     Normal         1.00      1.00      1.00     4572206
22         SQL         0.93      0.86      0.89         789
23         SSDP         1.00      1.00      1.00     1649955
24         SSH         0.94      0.79      0.86         3565
25 WebsiteSpoof       0.99      0.98      0.98      121533
26
27      accuracy                   1.00     6404584
28      macro avg           0.96      0.89      0.92     6404584
29      weighted avg          1.00      1.00      1.00     6404584
30
31 Confusion Matrix
32 [[ 13082      17    3957         0         0         2         2]
33 [      17   32454    6994         0         0        10         1]
34 [      649    3821  4565771        51         2       161      1751]
35 [         0         0     113      676         0         0         0]
36 [         0         0         2         0  1649953         0         0]
37 [         5        20     707         0         0      2833         0]
38 [         4         0     2723         0         0         0    118806]]
```

E.2.3 RF Model ID 1 - Raw Metrics

```
1 S-CV Results
2 Mean AUC = 99.99
3 Mean F1 = 99.66
4 Mean Precision = 99.66
5 Mean Recall = 99.67
6 Mean Accuracy = 99.67
7 Training Time 7794.549654006958 seconds
8
9 Final Test Results
10 Test AUC: 0.9999070506312879
11 Weighted Test F1: 0.996638797834701
12 Weighted Test Precision: 0.9966379719195173
13 Weighted Test Recall: 0.9967196932696956
14 Test Accuracy: 0.9967196932696956
15
16 Classification Report
17           precision    recall  f1-score   support
18
19      Botnet           0.95      0.77      0.85      17060
20      Malware           0.89      0.82      0.86      39476
21      Normal           1.00      1.00      1.00     4572206
22      SQL              0.93      0.86      0.89         789
23      SSDP             1.00      1.00      1.00     1649955
24      SSH              0.94      0.79      0.86         3565
25 WebsiteSpoof         0.99      0.98      0.98      121533
26
27      accuracy                   1.00     6404584
28      macro avg           0.96      0.89      0.92     6404584
29      weighted avg        1.00      1.00      1.00     6404584
30
31 Confusion Matrix
32 [[ 13082      17   3957         0         0         2         2]
33 [      17  32454   6994         0         0        10         1]
34 [      649   3821 4565771        51         2       161      1751]
35 [         0         0    113       676         0         0         0]
36 [         0         0         2         0 1649953         0         0]
37 [         5        20       707         0         0      2833         0]
38 [         4         0      2723         0         0         0 118806]]
```

E.2.4 RF Model ID 2 - Raw Metrics

```
1 Final Test Results
2 Test AUC: 0.9999070506308619
3 Weighted Test Precision: 0.9966379719195173
4 Weighted Test Recall: 0.9967196932696956
5 Weighted Test F1: 0.996638797834701
6 Test Accuracy: 0.9967196932696956
7
8 Classification Report
9           precision    recall  f1-score   support
10
11      Botnet           0.95      0.77      0.85      17060
12      Malware           0.89      0.82      0.86      39476
13      Normal           1.00      1.00      1.00     4572206
14  SQL_Injection         0.93      0.86      0.89         789
15      SSDP             1.00      1.00      1.00     1649955
16      SSH              0.94      0.79      0.86         3565
17  Website_spoofing      0.99      0.98      0.98      121533
18
19           accuracy
20      macro avg           0.96      0.89      0.92     6404584
21      weighted avg        1.00      1.00      1.00     6404584
22
23 Confusion Matrix
24 [[ 13082    17    3957         0         0         2         2]
25  [    17   32454    6994         0         0        10         1]
26  [    649    3821  4565771        51         2       161      1751]
27  [         0         0    113       676         0         0         0]
28  [         0         0         2         0  1649953         0         0]
29  [         5        20    707         0         0      2833         0]
30  [         4         0   2723         0         0         0   118806]]
```

E.2.5 RF Model ID 3 - Raw Metrics

```
1 S-CV Results
2 Mean AUC = 0.9999
3 Mean F1 = 0.9966
4 Mean Precision = 0.9966
5 Mean Recall = 0.9967
6 Mean Accuracy = 0.9967
7 Training Time 56042.87267756462 seconds
8
9 Final Test Results
10 Weighted AUC: 0.9999070506308619
11 Weighted F1: 0.996638797834701
12 Weighted Precision: 0.9966379719195173
13 Weighted Recall: 0.9967196932696956
14 Accuracy: 0.9967196932696956
15
16 Classification Report
17           precision    recall  f1-score   support
18
19      Botnet           0.95      0.77      0.85      17060
20      Malware           0.89      0.82      0.86      39476
21      Normal           1.00      1.00      1.00     4572206
22      SQL_Injection     0.93      0.86      0.89         789
23      SSDP              1.00      1.00      1.00    1649955
24      SSH               0.94      0.79      0.86         3565
25      Website_spoofing  0.99      0.98      0.98     121533
26
27      accuracy                   1.00    6404584
28      macro avg           0.96      0.89      0.92    6404584
29      weighted avg        1.00      1.00      1.00    6404584
30
31
32 Confusion Matrix
33
34 [[ 13082      17   3957      0      0      2      2]
35 [      17  32454   6994      0      0     10      1]
36 [      649   3821 4565771     51      2    161   1751]
37 [         0         0    113    676         0         0]
38 [         0         0         2         0 1649953         0         0]
39 [         5        20     707         0         0    2833         0]
40 [         4         0    2723         0         0         0 118806]]
```

E.2.6 RF Model ID 4 - Raw Metrics

```
1 S-CV Results
2 Mean AUC = 99.95
3 Mean F1 = 95.23
4 Mean Precision = 98.50
5 Mean Recall = 92.96
6 Mean Accuracy = 92.96
7 Training Time = 10147 seconds
8
9 Final Test Results
10 Weighted AUC: 0.9994792868436975
11 Weighted Precision: 0.984757273176817
12 Weighted Recall: 0.925409987596384
13 Weighted F1: 0.9496738038716113
14 Accuracy: 0.925409987596384
15
16 Classification Report
17
18                precision    recall  f1-score   support
19
20     Botnet           0.14       0.96       0.24       17060
21     Malware           0.22       0.97       0.36       39476
22     Normal           1.00       0.90       0.95      4572206
23  SQL_Injection       0.01       0.99       0.02         789
24         SSDP           1.00       1.00       1.00     1649955
25         SSH           0.04       0.99       0.08         3565
26  Website_spoofing    0.61       0.98       0.75      121533
27
28         accuracy                0.93     6404584
29         macro avg           0.43       0.97       0.48     6404584
30         weighted avg        0.98       0.93       0.95     6404584
31
32
33 Confusion Matrix
34
35 [[ 16326     90     30     91     0     504     19]
36 [    76   38392     13    170     0     824     1]
37 [ 102163  135709 4098890   76381     2   83351   75710]
38 [     0         0         1    785     0         3         0]
39 [     0         0        10         0 1649945         0         0]
40 [     6        15         4        16         0    3523         1]
41 [    282    1145    484    262         0     355   119005]]
```

E.2.7 RF Model ID 5 - Raw Metrics

```
1 S-CV Results
2 Mean AUC = 0.9987
3 Mean F1 = 0.9153
4 Mean Precision = 0.9842
5 Mean Recall = 0.8665
6 Mean Accuracy = 0.8665
7 Training Time 4632.155310869217 seconds
8
9 Final Test Results
10 Weighted AUC: 0.998635554230961
11 Weighted Precision: 0.9848384599880898
12 Weighted Recall: 0.8600619493787575
13 Weighted F1: 0.9116563847511311
14 Accuracy: 0.8600619493787575
15
16 Classification Report
17
18           precision    recall  f1-score   support
19
20    Botnet           0.06       0.94       0.12       17060
21    Malware           0.10       0.90       0.19       39476
22    Normal           1.00       0.81       0.89      4572206
23     SQL            0.01       0.99       0.01         789
24     SSDP           0.99       1.00       1.00     1649955
25     SSH            0.02       0.99       0.04         3565
26    WebSpoof         0.76       0.92       0.83     121533
27
28    accuracy                   0.86     6404584
29    macro avg           0.42       0.94       0.44     6404584
30    weighted avg         0.98       0.86       0.91     6404584
31
32
33 Confusion Matrix
34
35 [[ 16064    141    26    133     0    693     3]
36 [    72  35584   121   125     0   3572     2]
37 [ 240322  301805 3690025 138640  11696 154013 35705]
38 [     0     0     0    783     0     5     1]
39 [     0     0     9     0 1649946     0     0]
40 [     6     1     0    12     0   3546     0]
41 [   4970   1711   301   1392     0    768 112391]]
```

E.3 XGBoost

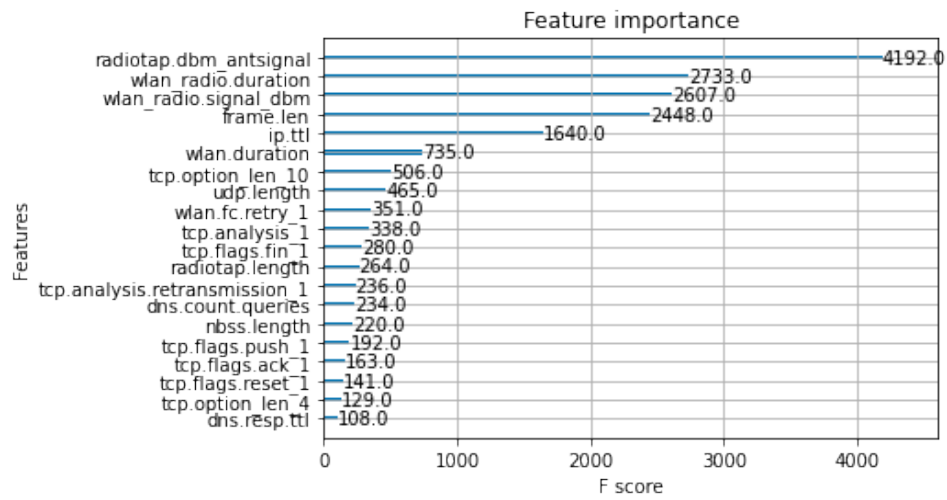
E.3.1 Stock 100% - XGBoost Raw Metrics

1	Final Test Results				
2					
3	Weighted AUC: 99.99				
4	Weighted F1: 99.65				
5	Weighted Precision: 99.65				
6	Weighted Recall: 99.65				
7	Accuracy: 99.65				
8					
9	Classification Report				
10					
11		precision	recall	f1-score	support
12					
13	0	0.96	0.74	0.84	17060
14	1	0.86	0.85	0.86	39476
15	2	1.00	1.00	1.00	4572206
16	3	0.93	0.88	0.90	789
17	4	1.00	1.00	1.00	1649955
18	5	0.95	0.79	0.86	3565
19	6	0.99	0.97	0.98	121533
20					
21	accuracy			1.00	6404584
22	macro avg	0.95	0.89	0.92	6404584
23	weighted avg	1.00	1.00	1.00	6404584
24					
25					
26	Confusion Matrix				
27					
28	[[12703 31 4320 0 0 2 4]				
29	[17 33596 5857 0 0 6 0]				
30	[539 5289 4564546 56 0 144 1632]				
31	[0 0 91 698 0 0 0]				
32	[0 0 0 0 1649955 0 0]				
33	[5 15 745 0 0 2800 0]				
34	[1 4 3344 0 0 0 118184]]				

E.3.2 Stock 100% - XGBoost CM



E.3.3 Stock 100% - XGBoost Feature Importance



F Neural Networks

F.1 MLP NN v1

```
1 # Create a sequential model
2 model = Sequential()
3 input_shape = (X_train_ohe.shape[1],)
4
5 # Add layers to the model
6 model.add(Dense(128, activation='relu', input_shape=input_shape))
7
8 model.add(Dense(64, activation='relu'))
9 model.add(Dense(7, activation='softmax'))
10
11 # Compile the model
12 model.compile(loss='categorical_crossentropy', optimizer='adam',
13               metrics=['accuracy'])
14
15 # Train the model
16 model.fit(X_train_ohe, y_train_ohe, epochs=10, batch_size=32,
17           validation_data=(X_test_ohe, y_test_ohe))
18
19 # Evaluate the model using test data
20 test_loss, test_acc = model.evaluate(X_test_ohe, y_test_ohe)
21
22 print('Test accuracy:', test_acc)
```

F.1.1 MLP Neural Network

```
1 from keras.models import Sequential
2 from keras.layers import Dense, Dropout, BatchNormalization
3 from keras.optimizers import SGD
4 from keras.initializers import he_uniform
5 from keras.metrics import AUC
6
7 # Define the number of classes
8 num_classes = 7
9
10 # Define the model architecture
11 model = Sequential()
12
13 # Add the input layer
14 model.add(Dense(100, input_shape=(X_train_ohe.shape[1],),
15                                activation='relu', kernel_initializer=he_uniform()))
16
17 # Add batch normalization
18 model.add(BatchNormalization())
19
20 # Add the first hidden layer
```

```

20 model.add(Dense(80, activation='relu', kernel_initializer=
    he_uniform()))
21 model.add(Dropout(0.25))
22 model.add(BatchNormalization())
23
24 # Add the second hidden layer
25 model.add(Dense(60, activation='relu', kernel_initializer=
    he_uniform()))
26 model.add(Dropout(0.2))
27 model.add(BatchNormalization())
28
29 # Add the third hidden layer
30 model.add(Dense(40, activation='relu', kernel_initializer=
    he_uniform()))
31 model.add(BatchNormalization())
32
33 # Add the fourth hidden layer
34 model.add(Dense(20, activation='relu', kernel_initializer=
    he_uniform()))
35 model.add(BatchNormalization())
36
37 # Add the output layer
38 model.add(Dense(num_classes, activation='softmax'))
39
40 # Define the optimizer
41 sgd = SGD(lr=0.01, momentum=0.9)
42
43 # Compile the model
44 model.compile(loss='categorical_crossentropy', optimizer=sgd,
    metrics=[AUC()])
45
46 # Train the model
47 batch_size = 170
48 epochs = 10
49 history = model.fit(X_train_ohe, y_train_ohe, batch_size=
    batch_size, epochs=epochs, validation_data=(X_test_ohe,
    y_test_ohe))
50
51 # Evaluate the model on your test data
52 test_loss, test_auc = model.evaluate(X_test_ohe, y_test_ohe)

```