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

In recent years, advancements in technology, such as machine learning, have seen a widespread increase in the reliance on computer systems for daily life. With this increased reliance, the complexity of cyber-attacks has increased. Conventional Intrusion Detection Systems (IDS) approaches have proven insufficient in detecting these emerging and advanced threats. Existing literature lacks the assessment of using machine learning in Wireless Network Intrusion Detection Systems (WIDS) to classify these using a combination of application layer features with 802.11 and non-802.11 network protocol features.

This project examines combining additional application layer features to train two ensembles (Random Forest & XGBoost) and one neural network based (MLP) machine learning model for a proposed WIDS. The benchmark Aegean Wi-Fi Intrusion Dataset 3 (AWID3) was used, and six attacks (Botnet, Malware, SQL Injection, SSH, SSDP Amplification and Website Spoofing) were chosen to be classified. Models were evaluated on metrics such as AUC, F1 and Cross-Validation scores. The range of models, without relying on data balancing techniques, demonstrated high classification performances in all AUC, F1, Precision, Recall and Accuracy metrics of up to 99.9%.

Keywords: Application Layer Attacks, AWID3 dataset, MLP, Random Forest, Wireless Network Intrusion Detection Systems, XGBoost

Abbreviations

Address Resolution Protocol	ARP
Area Under Curve	AUC
Aegan Wi-Fi Intrusion Dataset v3	AWID3
Autoencoders	AE
Categorical Crossentropy	CC
Comma-Separated Values	CSV
Deep Neural Network	DNN
Denial Of Service	DoS
Distributed Denial of Service	DDoS
Domain Name Service	DNS
F-Score/F-Measure	F1
Hypertext Transfer Protocol	HTTP
Intrusion Detection System	IDS
K Nearest Neighbour	KNN
Machine Learning	ML
Man-in-the-middle	MITM
Multi-Layer Perceptron	MLP
Neural Network	NN
One-Hot Encoding	OHE
Protected Management Frames	PMF
Random Forest	RF
Secure Shell	SSH
Server Message Block	SMB
Simultaneous Authentication of Equals	SAE
Simple Service Discovery Protocol	SSDP
Stochastic Gradient Descent	SGD
Stratified Cross Validation	S-CV
Transmission Control Protocol	TCP
User Datagram Protocol	UDP
Wireless Network Intrusion Detection System	WIDS
eXtreme Gradient Boosting	XGBoost

Contents

List of Figures

List of Tables

1 Introduction

The ongoing increase in IoT devices in homes and commercial environments has seen a surge in wireless networks, particularly IEEE 802.11 networks, commonly referred to as Wi-Fi. As businesses and consumers seek to try out these new devices and technologies, manufacturers tend to prioritise improving performance and features, neglecting 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 within a network.

1.1 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 Evil Twin attacks (Sivalingam 2021). 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 (Vanhoef 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 PMFs and SAE authentication methods have shortcomings, including being vulnerable to denial-of-service, side-channel, and downgrade attacks (Vanhoef and Ronen 2020).

1.2 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 initially feeding the system with a large amount of data to learn an environment's regular usage patterns.

External tools such as Stratosphere IPS (SLIPS) developed by Garcia, Gomaa, 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 investigating and developing wireless intrusion detection systems using machine learning-based algorithms utilising supervised, unsupervised and deep learning approaches in 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 wireless intrusion detection system.

1.3 Research Questions and Objectives

The objectives for the project are as follows:

- To explore and analyse current literature and academic research utilising machine learning 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, proving 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 for detecting network attacks on 802.11 wireless networks. The following papers and literature inspire the practical element of this project.

2.1 Datasets

Hnamte and Hussain (2021) discusses 37 public datasets, their suitability for building and training an IDS, and their 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, public network traffic ranging from regular user activity to 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 UNSW-NB15 dataset, (Moustafa and Slay 2015) created by The University of New South Wales in Australia, is a well-known network intrusion detection dataset consisting of 49 features with nine attack classes, specifically: Analysis, Fuzzers, Worms, DoS, Reconnaissance, Generic, Exploits, Shellcode and Backdoors. It seeks to replace older datasets such as KDD98, KDDCUP99, and NSLKDD, frequently used to evaluate NIDS. However, the dataset was generated on non-wireless hardware and therefore did not align with the requirements of a wireless network dataset.

The recently published 5G-NIDD dataset (Siriwardhana 2022) presents a labelled dataset built using 5G networks and contains a series of attack scenarios such as DDoS and port scans. As a relatively new dataset, it lacks existing literature and research for its utilisation for training an IDS. Moreover, being generated on 5G networks, it fails to meet this project's requirements of needing an 802.11w network dataset.

The AWID3 dataset (Chatzoglou, Kambourakis, and Kolias 2021) released in February 2021 seeks to build upon the existing AWID2 dataset 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 (Kolias et al. 2016). The dataset includes the Pairwise Master Key (PMK) and TLS Keys. Additionally, AWID3's concentration on enterprise networks includes Protected Management Frames (PMF) that help 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 dataset (ibid.), 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 concerning application layer attacks, AWID3 is a strong candidate for investigating the development of an IDS using machine learning.

2.2 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 successfully identified the attack with an average detection time of 3.42 seconds.

2.3 Detecting Network Attacks

Application Layer Attacks

Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos (2022a) discusses detecting 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 because these are usually encrypted and, therefore, not easily accessible. Moreover, it raises privacy concerns, 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)). Bagging produced the highest-scoring AUC of the classifiers, 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. When the two feature sets were combined, the AUC saw a score of up to 95.30%. Additionally, an 'insider feature' was engineered to represent if packets in the Botnet class are sent via client-client or client-server. This feature saw an improvement of up to 3% in LightGMB and Bagging models. It is clear that this paper does not address the problem of using a set of application features or any optimisation techniques.

5G Attacks

Mughaid et al. (2022) discusses the rise and need for protection from 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 proposes a new system that leverages machine learning and deep learning techniques to achieve a high detection accuracy. 99% accuracy was achieved using KNN and 93% with DF and Neural Network.

Attack Classifications

Islam and Allayear (2022) utilised the AWID dataset to predict tuples 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 on the testbed's Access Point were vulnerable, and a chosen Intrusion Detection System could not identify and detect the attacks. Dalal et al. (ibid.) then designed 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 uses logical reasoning to deduce an attack rather than anomaly detection, such as machine learning.

Saini, Halder, and Baswade (2022) investigated the detection of WPA3 attacks in the context of intrusion detection using their curated dataset based on existing and known WPA3 attacks: De-Authentication, RogueAP, Evil Twin, Krack and Beacon Flooding. A two-staged intrusion detection system is proposed. Traffic is first run through a Flood Detection system at the AP to detect sudden surges of data packets and secondly using an ML-based classifier. The data was trained on Logistic Regression, Decision Tree and Random forest and achieved a high accuracy of 99.98% on Decision Tree and 99.97% on Random Forest.

Uszko et al. (2023) utilised the AWID3 dataset for a proposed IDS for anomaly detection; the data was selected based on frames, and an equal number of frames were selected per class. Of the models tested, Decision Tree and Naive Bayes performed the best. Decision Tree achieved the best results with 98.57% accuracy on the validation set and 96.79% and 97.03% accuracy on the custom testbed created with Beacon Flood and de-authentication attacks. The paper addresses the common issue of testing the created models on a data environment different from training.

In Liu et al. (2022) proposed an IDS capable of detecting DoS attacks on wireless sensor networks. Using the WSN-DS dataset, the K-Nearest Neighbor classifier was implemented with an arithmetic optimisation algorithm (AOA), and additionally, the Levy flight strategy was used for optimisation adjustment. The experiments concluded that the model reached up to 99% accuracy, nearly a 10% improvement from the original KNN algorithm.

The works by Torrres (2021) utilised KNN on the AWID2 & 3 datasets on ten features. 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 Machine Learning Algorithms

The following table summarises a selection of existing literature and papers from the past five years related to the use of machine learning in detecting network attacks. The following common algorithms are abbreviated as follows: Random Forest (RF), Decision Tree (DT), Multi-Layer Perceptron (MLP), AutoEncoder (AE), Logistic Regression (Logreg), Neural Network (NN), Support Vector Machine (SVM), Naïve Bayes (NB) and K-Nearest Neighbour (KNN). It concludes that a wide array of machine learning algorithms have been utilised to detect network attacks. However, gaps still remain in using Random Forest and XGBoost on the AWID3 dataset.

Table 2.1: Existing Literature Using ML Techniques

Work	**Dataset/Data**	**ML Methods**
Lopez Perez et al. 2018	MSU Scarda Dataset	SVM, RF
Ge et al. 2019	Bot-IoT	Feed-Forward NN
Ran, Ji, and Tang 2019	AWID2	Ladder Network
Smys, Basar, Wang, et al. 2020	UNSW NB15	Hybrid Convolutional Neural Network
Kachavimath, Nazare, and Akki 2020	KDDCup99, NSL-KDD	KNN, NB
Satam and Hariri 2021	AWID2 & University of Arizona Dataset	IsolationForest, C4.5, RF, AdaBoost, DecisionTable
Dalal et al. 2021	Mininet 2.2.2	SVM, MLP, DT, RF
Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos 2022a	AWID3	DT, LightGBM, Bagging, MLP & AE
Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos 2022b	AWID 2 & 3	Logreg, SGDClassifier, LinearSVC, LightGBM, DT, RF, Extra Trees, MLP, AE
Saini, Halder, and Baswade 2022	AWID3	Logreg, DT, RF
Islam and Allayear 2022	AWID3	KNN
Mughaid et al. 2022	AWID3	DT, KNN, Decision Jungle, Decision Forest, Neural Network
Liu et al. 2022	WSN-DS	KNN
Dhanya et al. 2023	UNSW-NB15	RF, AdaBoost, XGBoost, KNN, MLP
Uszko et al. 2023	AWID3	DT, NB, RF, MLP
Agrawal, Chatterjee, and Maiti 2022	AWID3	XGBoost, LightGBM, CatBoost
Le, Oktian, and Kim 2022	X-IIoTDS, TON_IoT	XGBoost
A. Reyes et al. 2020	AWID2	Bagging, RF, ET, XGBoost, NB

2.5 Summary

Based on the literature review and research on the AWID3 dataset and wireless network attack classification, detecting application layer wireless network attacks using machine learning is under-researched. In their previous work, Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos (2022a) 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 the method of attack individually, 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 the AWID3 dataset.

3 Methodology

3.1 Ethics & Risks

Ethical approval was not required for this project and can be found in A. The following risks and ethical concerns are addressed as follows:

- Data Reliability and Quality: Public datasets may vary in data quality and can lead to possibly unreliable results and conclusions. The chosen dataset is well-established and has extensive existing literature and research.
- Privacy Concerns: Datasets may contain personally identifiable information; however, in the context of this project and AWID3, features that may contain personal information will not be used for this project.

In summary, no significant risks were identified, and no mitigations are required for this project.

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 Apple Silicon limitations and errors encountered, TensorFlow GPU Acceleration was not utilised for Deep Learning on the M2 Mac Mini.

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 complete 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 Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos (2022a), six attacks out of the 13 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.

The following details relevant background information about each attack class (Kolias et al. 2016).

- SSH Bruteforce - a brute-force attack was conducted against the radius server unsuccessfully for 180 seconds on the login credentials.
- Botnet - The attack deployed pieces of malware within a Samba shared directory and assumed victims executed them. Four STAS were then infected, turning into bots. Remote commands were then executed, such as grabbing a screenshot of the desktop and sent to the attacker.
- Malware - Two pieces of malware were placed within a Samba share and downloaded by six STAs, but never executed.
- SQL Injection - The target is an external node (DVWA), and a malicious SQL query string was inputted into a web form of the target. The packet's HTTP POST and GET requests can reveal the SQL code query.
- SSDP Amplification - This attack consists of a DDoS attack using the Simple Service Discovery Protocol. It uses Universal Plug and Play (UPnP) to trick all STAs of the wireless network into sending a barrage of packets to each SSDP-enabled device. Every device then responds, eventually leading to a DoS. In the dataset, the attacker scanned the intranet for 30 seconds before launching the attack for 210 seconds on the DVWA webpage.
- Website Spoofing - The attack deployed a fake Instagram webpage and used ARP and DNS poisoning to redirect victims to the fake page, where entered credentials were stolen and decrypted.

This work aims to combine the (16) 802.11 and (17) non-802.11 features from Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos (2022a) with a set of chosen application layer features to detect and classify the different application layer attacks. As previously established, existing research determined a high degree of accuracy and performance when combining the 802.11 and non-802.11 features. Still, there is a lack of research into whether including additional application layer features would provide grounds for 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 attack CSV file, 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 organisation'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, these features were not selected for this study to ensure data privacy and avoid bias from information specific to the AWID3 environment or containing identifiable information such as URLs and IP addresses. 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 (13)		
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
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.ttl	Min-Max	int64
ssh.packet_length	Min-Max	int64

Table 3.1: The selected set of application layer features.

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

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 particular activities, such as file transfers and remote execution etc. The packets' length can also help identify any anomalous activity 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 received. These HTTP features can help identify potential attacks exploiting web-based vulnerabilities such as SQL Injections or Website Spoofing.

Domain Name System (DNS) converts 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 help identify potential reconnaissance attacks and provide insights into the network traffic patterns to identify possible 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 abnormal behaviour indicative of an attack.

Initially, 19 application layer features were chosen to be included in this research; however, in the later stages of data preprocessing, six features, as seen below, were dropped as they were found to contain zero values for the selected attack classes. Therefore, it was concluded to exclude these features.

- nbss.type
- smb.access.generic_execute
- smb.access.generic_read
- smb.access.generic_write
- smb.flags.response
- dns.resp.len

3.4.2 802.11 and Non-802.11 Features

The selection of 802.11 and non-802.11 features was chosen primarily based on the findings of two previous literature papers, (Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos 2022b; Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos 2022a) which demonstrated high performance and results in similar research and work in the same problem domain. Consideration was made to carry out feature selection. Still, by building on the proven success of existing literature, this project aims to help contribute and validate the claims made by the selected features in the context of detecting network attacks on the AWID3 dataset.

Table 3.2 and Table 3.3 show the 802.11 and non-802.11 features used for this project. The wireless-specific features, such as the radio signal strength, duration, frame lengths etc., have proven to be important in detecting attacks, as observed from the previous papers. It consists of Transport layer (TCP & UDP) protocol features responsible for data transfer, and ARP features that operate on the Data-link layer to resolve Mac addresses.

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.pwrmgt	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.

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 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. 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 multiple 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 follows:

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.

Combining 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 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 while importing 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, forming one data frame. This saw a reduction in import time and lower memory consumption.

Data Cleaning

The data was cleaned to ensure it was fit for the next data preprocessing stage. Rows that contained only NAN values were dropped, as well as missing Label values. Each column's missing/nan values were replaced and represented with 0, following a similar approach to Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos (2022a).

Upon analysis, frequent hyphened values were observed, e.g. *-100-100-10, 123-456-1, -10-2, 81-63-63 etc.*. 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 proposed 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 complete code for this section can be found in Appendix B.2.

Individual Datasets

After data cleaning and processing, the final six 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 the 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, One-Hot Encoding were used 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. Some algorithms are sensitive to the scale and may put more importance on certain features if not scaled. MinMax 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

The dataset is imbalanced at its core, with most '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 typical 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 model's performance to help prevent overfitting. Overfitting can occur when the model learns all of the training data's features and relationships, 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
SSDP	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 significant 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 at 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 Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos (2022a). The training set is split into ten folds; the model is then trained on all folds except the validation set. The model is then tested on the validation set for its performance metrics and recorded. This is repeated for all ten folds, so each 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 entire training set and evaluated based on the testing set to obtain a final performance measure before saving the model.

3.10 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. A review of existing literature and research in Section 2 shows that a wide range of machine learning algorithms has been used for the purposes of classifying network attacks. There exists a research gap in the unexplored machine learning algorithms. As such, this study aims to explore the effectiveness of a few algorithms. AWID3 is a labelled dataset; as such, only supervised algorithms were used for this work. The following algorithms were employed due to their effectiveness in existing literature, reproducibility and the identified research gap in current research regarding their application to this specific task. These algorithms have been established and shown success in other machine-learning tasks and have been adopted within the existing literature; their implementation is readily available from well-known ML libraries such as TensorFlow, Sci-Kit learn and XGBoost (Pedregosa et al. 2011; Martín Abadi et al. 2015; Chen and Guestrin 2016). The models were coded using prior module knowledge and relevant libraries' documentation.

3.10.1 Random Forest

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

3.10.2 K-Nearest Neighbor

K-Nearest Neighbor is a non-parametric algorithm that finds the k-closest neighbours to a given input. It classifies it based on the majority class within the k neighbours from a chosen metric, for example, the Euclidean distance. It is considered a more computationally intensive algorithm, requiring observing the training data during evaluation to make predictions.

3.10.3 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 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.

3.10.4 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 connected to additional layers through weighted connections. During training, the gradient of the loss function (difference between the predicted values with the actual values) is calculated and the weights and biases are updated with an optimiser to ensure the model is able to generalise and learn from the data.

3.11 Evaluation Metrics

A vital area of the work was deciding the use of specific metrics to evaluate the performance of the models. Metrics are essential 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 models' performances. 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 essential to minimise false negatives and false positives, which represent misclassifying the attack on the network traffic as either a potential intrusion (false negative) or falsely marking regular traffic as malicious (false positives). By 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 distinguish between positive and negative classes correctly. AUC-ROC is also insensitive to class imbalances. Similarly, in the works carried out in (Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos 2022a; Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos 2022b), 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 measures the proportions of positive values that were correctly classified. Similarly, the FPR is the proportion of negative values that are incorrectly classified as positive. The area under the curve (AUC) is calculated using the ROC curve. This value ranges between 0 and 1, where 0.5 represents, at best random guessing, and one corresponds to a perfect classifier.

As the 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.

Equations for Precision, Recall & F1

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

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

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

Micro, Macro and Weighted

In regular binary classification, metrics such as F1, Precision, Recall and AUC can be calculated easily; however, for multi-class classification problems, 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 consider 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 typically used when all classes are equally important, regardless of class size or imbalance.
- 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 consider 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, as underperforming classes can be identified, allowing guidance for tuning and modifications.

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 wrongly predict each class for another (confusion). By examining the confusion matrix, any specific classes that may require additional tuning or changes to the model can be identified. 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 confusion matrix's importance in imbalanced datasets.

Feature Importance

Feature Importance is a metric that determines the relative importance of each feature in predicting the output. XGBoost and Random Forest, being ensemble learning algorithms, provide feature importance scores. The top 20 features in each model will be plotted in a graph. This metric can provide insights into model interpretation and domain understanding of the problem and which features have a higher impact, helping select features.

4 Experiments

This project utilised a diverse set of machine-learning algorithms for this problem. Consisting of three shallow classifiers: Random Forest, K-Nearest Neighbour and XGBoost and one neural network: Multi-Layered Perceptron. The code for all models and experiments can be found within the codebase and Appendix G.

4.1 Initial Modelling

To speed up initial training and testing for each machine learning algorithm, many subsets of the original combined data were created using sklearn's train_test_split to create a stratified split resulting in reduced datasets. Varying data splits were made, 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 each model's suitability and help provide a rough measure of a model's underlying data performance. As discussed previously, additional Cross Validation is used during training where appropriate.

4.2 Parameter Tuning

An important aspect of experimentation is tuning parameters specific to each model. Hyperparameters are defined during the creation of each model and can substantially impact its performance. As such, two primary methods are adopted. Sometimes, 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. However, this can be computationally expensive and time-consuming, so 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, iterative experimentation and domain knowledge were 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

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 prior knowledge and iterative testing to create a series of models that were subsequently evaluated and optimised. The training was conducted on both the M2 and VM testbeds. Table 4.1 details the 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 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 essential to ensure modifications made to the model avoided overfitting.

Subsequent experimentation was conducted using the model's training metrics, confusion matrices, classification reports and predictions on the test set. Models 3 and 5 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 higher recall and very low precision rates for several classes.

During 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 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 terminated at this stage.

Table 4.1: Parameters for Random Forest Classifier (Pedregosa et al. 2011)

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 using 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. However, in Model 3, early stopping and regularisation were added, with no noticeable performance increase/decrease 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.

Initial experimentation with parameter searching on a smaller 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 consistent across the ten folds.

See Listing 1 for the parameter grids used.

```

gscv_param_grid = {
    'learning_rate': [0.05, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5]
}

rgs_param_grid = {
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [3, 6, 9],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.2],
    'subsample': [0.5, 0.7, 0.9],
    'colsample_bytree': [0.5, 0.7, 0.9],
    'n_estimators': [100, 200]
}
```

Listing 1: Grid Search Parameters For XGBoost

Table 4.4: XGBoost GridSearch Best 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

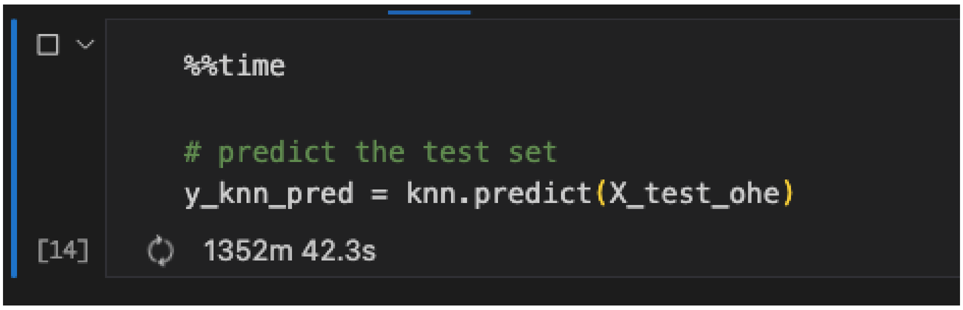
The parameter tuning process helped to identify a set of optimal parameters using RandomisedSearchCV, as detailed in Table 4.4. These parameters were used to create Model 10. Subsequent models made afterwards, from additional parameter tuning, did not show a noticeable improvement in the model's performance.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

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 data test set and 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 keeps them in memory. As a result, predicting on the test set 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 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



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. A wide range of MLP models was explored, and 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 parameters used for each notable MLP model.

Experimentation began with a three-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 exact parameters tested across the 60 and 80% datasets; metrics were high. However, the models struggled to predict minority classes, resulting in low recall and F1-Scores. Performance when increasing the size of the dataset did improve the performance. It can be attributed to the fact the larger dataset provided more samples of the minority class to be trained on. This information was used to create further models 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 most 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 generalise). The model would stop training once the validation loss began to degrade over two defined epochs. Dropout is a regularisation method that randomly sets 0.2 of the input neurons to 0. Dropout layers were used in the network architecture.

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.

Thresholds

Towards the latter stages of experimentation, further attempts were made to enhance the models' performance 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 prediction changes. However, this was not explored with great depth, leading the door for future work.

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 many experiments and achieving high-performance results, the decision was made to conclude further model experimentation.

Table 4.5: MLP Model Parameters Pt 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 Pt 2

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 models on the test set is analysed and discussed. The models are evaluated using previously discussed metrics, 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 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 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 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 SSDP, 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; the confusion matrix further verifies this. Models 1 and 2 were trained on the entire dataset, with the exception that model 1 was trained with 10 Fold S-CV; 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 CV and Testing, suggesting the models are not overfitting. As seen in model 0, the classification reports and confusion matrix share similar performances, with models 1 and 2 having fewer misclassifications, i.e., from 7 false positives to 2 on the SSDP class. This may indicate that despite adding more data to training, it 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 the classifier to handle imbalanced datasets; its default value is *None*, meaning the model treats every class equally 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 contain 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 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 across most models' top five features.
- *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 to the other models. Future work should focus on using GridSearchCV and Randomised-SearchCV to provide more advanced parameter tuning. Moreover, further emphasis can be placed on the minority classes to help lower the number of false positives.

5.2 XGBoost

Table 5.3 summarises the average metrics with 10 Fold S-CV, 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, allowing 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 was 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 minor differences but did not significantly affect performance. Adding early stopping and regularisation to model 3 did not significantly affect the model's performance compared to the baseline.

GridSearchCV

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 five times and took 28.27 hours to complete. Models 5 (80%) and 8 (100%) were created with these parameters. The results are incomparable compared to their baseline counterparts (4 and 6). 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 those 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 high metrics and behaviour.

RandomisedSearchCV

A parameter grid was tested with RandomisedSearchCV and took 41.81 hours to complete. Each parameter combination was subjected to 10-fold Stratified Cross Validation to measure its performance.

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 performs well for most classes, especially Normal, SSDP Amplification 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%
- WebSpoof: 3046 / 121533 = 2.5%

Feature Importance

The feature importance shows the features that have the biggest impact on the model's predictions; from the models combined, a few features ranked consistently with the XGBoost classifiers: *ARP, ip.ttl, arp.hw.size, tcp.checksum.status, http.request.method, http.response.code, http.content_type*. These common features existed in the top 20 features across all models, and the three application layer features from HTTP also significantly impacted the models.

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 the baseline parameter model could still deliver remarkably similar results despite this. Future investigations may consider a bigger search grid with more focused tuning based on the specific characteristics of the classes or tasks.

5.3 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 several hidden layers and neurons, optimisers (Adam & SGD), regularisation techniques (Dropout and Early Stopping), learning rates etc. 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

Epoch & Batch Size

The number of epochs is the number of complete passes the model will be trained on in the dataset. When the number is too low, the model may fail to learn the data and its relationships and under-fits, performing poorly on unseen data. Alternatively, when the number of epochs is too high, the model may overfit and memorise the training data, performing poorly on new data. The batch size defines the number of samples the model works through before the model's internal parameters are changed (Brownlee 2018).

Model 4 highlights the importance of dataset size and parameters such as batch size and epochs; However, the model shares similar parameters as Models 0-3; the batch size increased from 180 to 200 and 15 epochs to 20. It is important to note that although the number of epochs was increased, it was observed during CV that the model would often stop at around ten epochs due to early stopping. The model delivered an outstanding performance on the test set with an AUC, F1, Precision, Recall and Accuracy all around 99%. In previous models, SQL Injection was predicted poorly, with low overall recall scores. The larger dataset and batch size helped increase this score, but the models generally struggle to classify this class correctly.

Data Subsets

Both 60% and 80% models showed high precision and recall when examining the results across the different data subsets. Class-specific performances for minority classes were consistently low. Notable, in model 3 on the 80% subset, recall increased 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 could reduce some false positives at the high cost of failing to identify the true positives.

Previous Works

The works by Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos (2022a) similarly used an MLP model consisting of four hidden layers; these specifications were adopted again in Model 7 to provide context for comparison. Although fundamentally different, their models achieved metrics of around 75% in AUC and 70% in F1 across the 802.11 and Non-802.11 sets. The model in this project 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 affirms this observation, with many predictions from those classes being misclassified as Normal traffic. This further emphasises the importance of tuning parameters and settings specific to the problem at hand.

Summary

The Multi-Layer Perceptron in TensorFlow can provide a vast array of parameters and options, leading to endless 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 many 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 balances complexity and constraints. To automate the process, further work can be investigated into utilising parameter searching such as GridSearchCV.

5.4 Comparison of Models

The purpose of this work serves to provide recommendations for developing a wireless network intrusion detection system; however, evaluating the performances of the models poses a challenge when determining the 'best' model for each algorithm. Challenges arise when multiple metrics and performance indicators are to be compared. The analysis focused on achieving a balance between avoiding false positives (instances where normal traffic is marked as malicious) and false negatives (instances of malicious traffic marked as normal). The aim was to identify a model capable of distinguishing between the six attack classes and avoiding as many false negatives and positives as possible. The following sections compare the three 'best' identified models from the following models: Random Forest, XGBoost and Multi-Layer Perceptron (MLP).

Random Forest

During experimentation, several Random Forest models were trained, and attempts were made to search through a series of parameters; using the evaluation metrics defined previously, Model ID 1 displayed robust and consistent performance during CV and testing. It achieved an AUC of 99.99, F1 of 99.66, Precision of 99.66, Recall of 99.67 and Accuracy of 99.67 on the test set, indicating that it was able to correctly classify the six attack classes with a high degree of accuracy. The model's confusion matrix shows a good balance between FP and FNs on majority classes; whilst it struggled slightly on Botnet and SSH, the total misclassification remained relatively low. 5.7, 5.1 and 5.2 show the Classification, Confusion Matrix and Feature Importances for the model.

Table 5.7: RF Model 1 - Classification Report

Class	Precision	Recall	F1-Score	Support
Botnet	0.95	**0.77**	**0.85**	17060
Malware	**0.89**	0.82	0.86	39476
Normal	1.00	1.00	1.00	4572206
SQL Injection	0.93	0.86	0.89	789
SSDP	1.00	1.00	1.00	1649955
SSH	0.94	0.79	0.86	3565
Website Spoofing	0.99	0.98	0.98	121533
Accuracy			1.00	6404584
Macro Avg	0.96	0.89	0.92	6404584
Weighted Avg	1.00	1.00	1.00	6404584

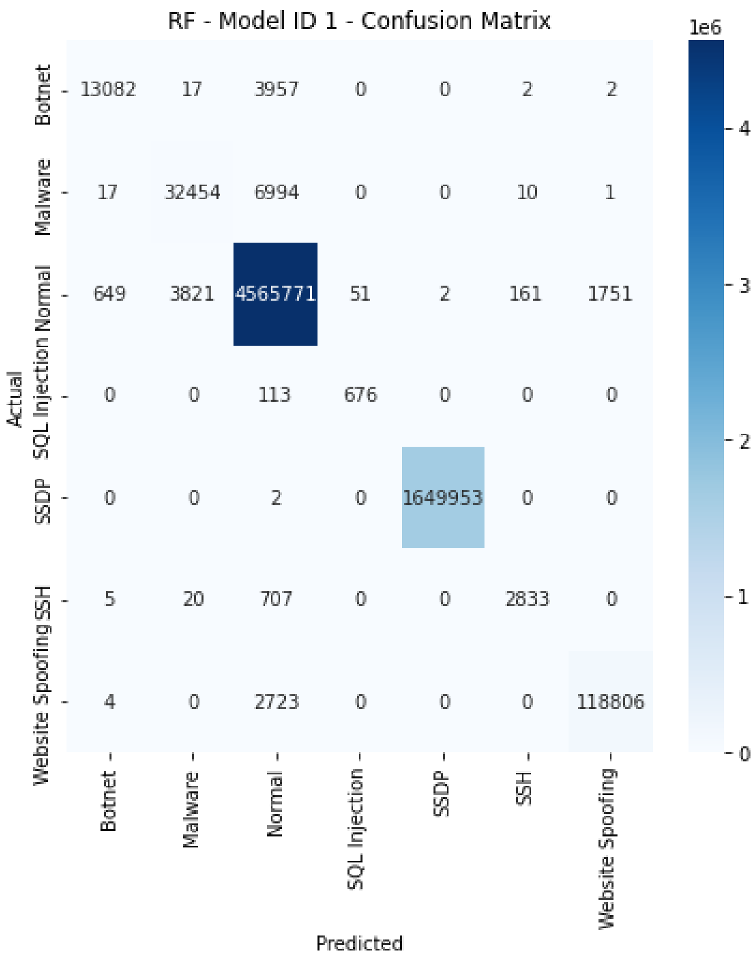


Figure 5.1: RF Model 1 - CM

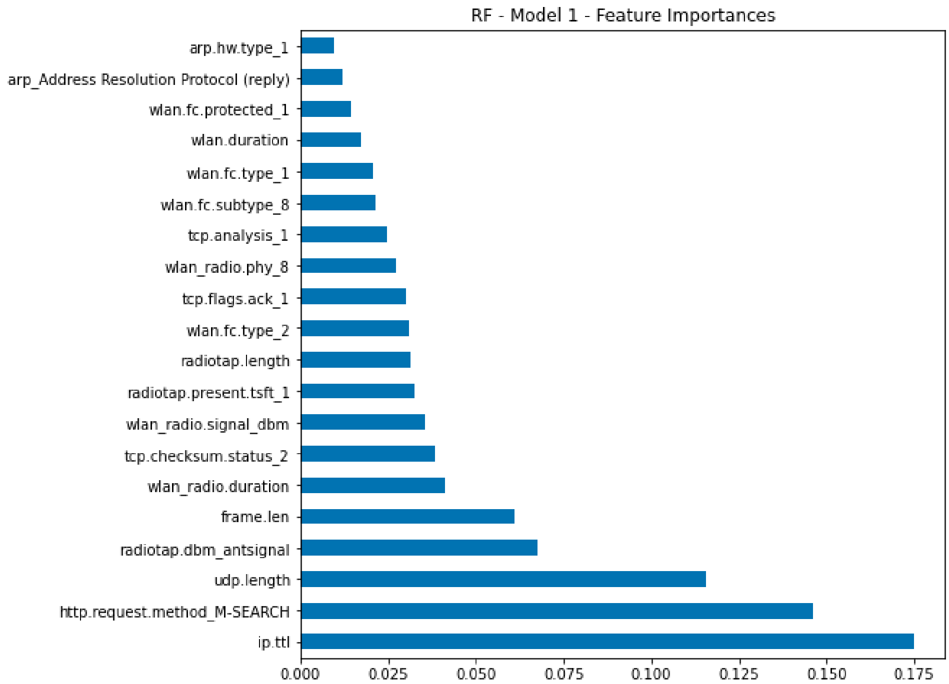


Figure 5.2: RF Model 1 - FI

XGBoost

Comparing all 11 models trained on the XGBoost Classifier, Model 10 achieved superiority with an AUC of 99.99 (rounding to 100), F1 of 99.65, Precision of 99.65, Recall of 99.66 and Accuracy of 99.66 across the test set and similar during Cross Validated training. It was considered to be the best-performing model from the selection and proved to be robust and efficient. 5.8, 5.3 and 5.4 show the Classification, Confusion Matrix and Feature Importances for the model.

Table 5.8: XGBoost Model 10 - Classification Report

Class	Precision	Recall	F1-Score	Support
Botnet	0.96	0.75	**0.84**	17060
Malware	**0.89**	0.82	0.85	39476
Normal	1.00	1.00	1.00	4572206
SQL Injection	0.94	0.89	**0.91**	789
SSDP	1.00	1.00	1.00	1649955
SSH	0.92	**0.78**	0.85	3565
Website Spoofing	0.99	0.97	0.98	121533
Accuracy			0.99	6404584
Macro Avg	0.83	0.73	0.80	6404584
Weighted Avg	0.99	0.99	0.99	6404584

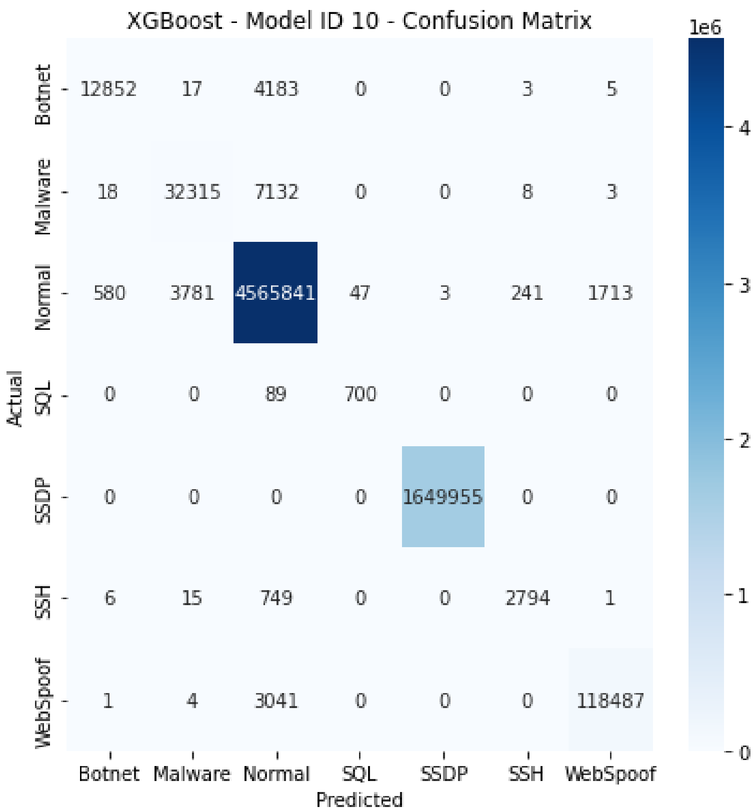


Figure 5.3: XGBoost Model 10 - CM

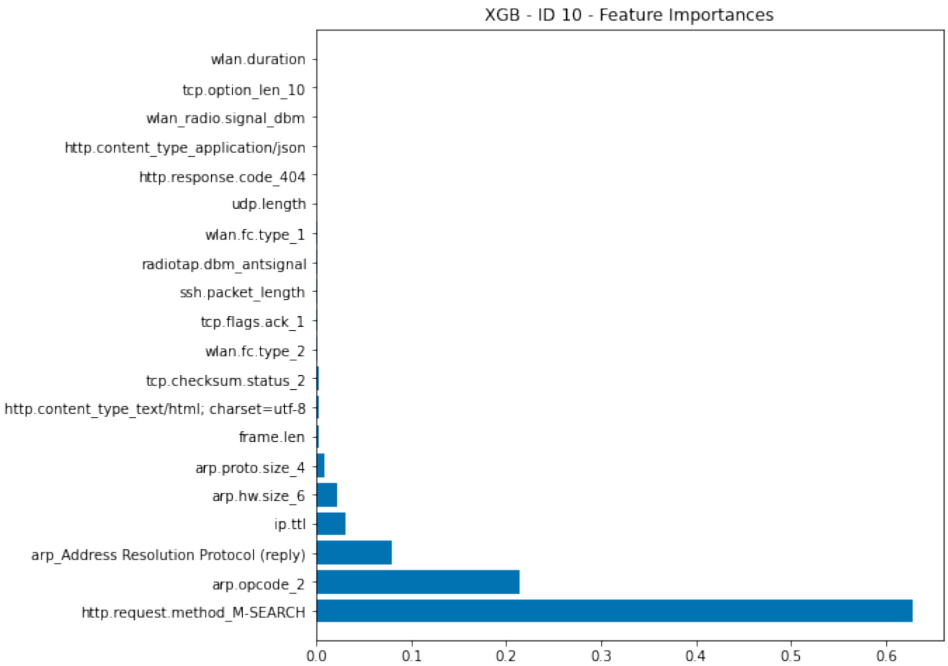


Figure 5.4: XGBoost Model 10 - FI

MLP

Out of all models trained, model 4 showed strong performance overall, with a good proportion of instances from classes Botnet, Malware, Normal and SSDP being accurately classified. While the model struggles with SQL Injection and SSH with poor recall and F1 scores, this was expected given the imbalance in the dataset. Metrics were consistent across S-CV and the test set, indicating that the MLP model was not overfitting the training data and generalising well on new unseen data. Refer to 5.9 and 5.5 for the Classification Report and Confusion Matrix for the model. The lowest scores per column are denoted in red. Figure 5.6 shows the best and worst fold between the ten folds in training. In the lowest fold, the model starts to overfit on the training data after epoch 2, and in two consecutive folds with no improvements, early stopping interrupted the fold.

Table 5.9: MLP Model 4 - Classification Report

Class	Precision	Recall	F1-Score	Support
Botnet	0.94	0.61	0.74	17060
Malware	0.89	0.72	0.80	39476
Normal	0.99	1.00	1.00	4572206
SQL Injection	0.99	**0.37**	**0.54**	789
SSDP	1.00	1.00	1.00	1649955
SSH	**0.83**	0.48	0.60	3565
Website Spoofing	1.00	0.92	0.95	121533
Accuracy			0.99	6404584
Macro Avg	0.83	0.73	0.80	6404584
Weighted Avg	0.99	0.99	0.99	6404584

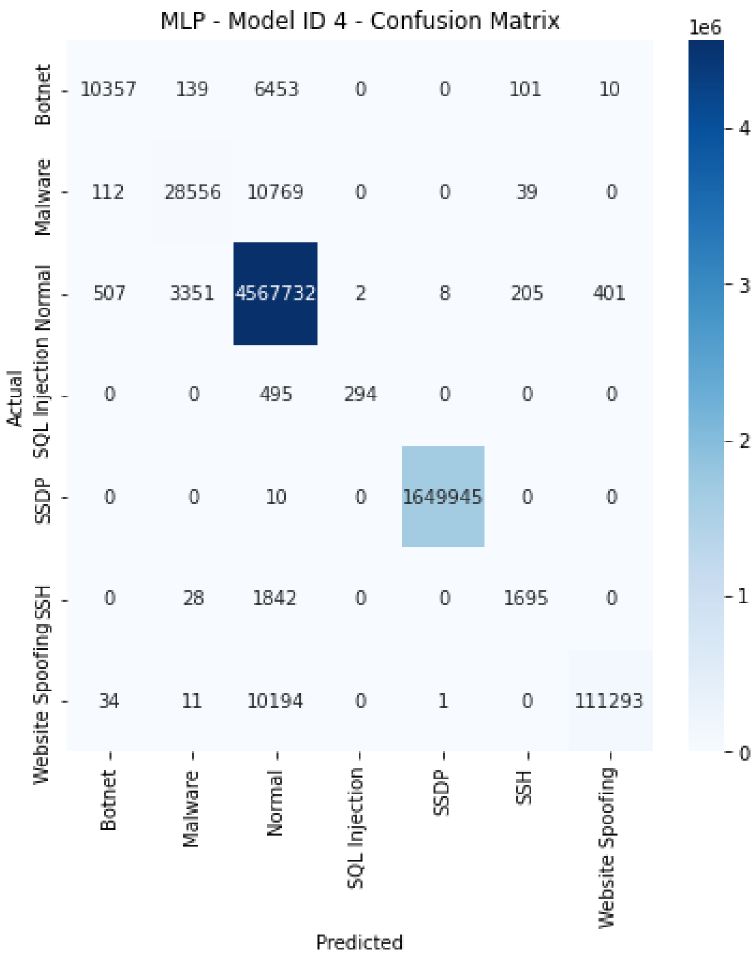
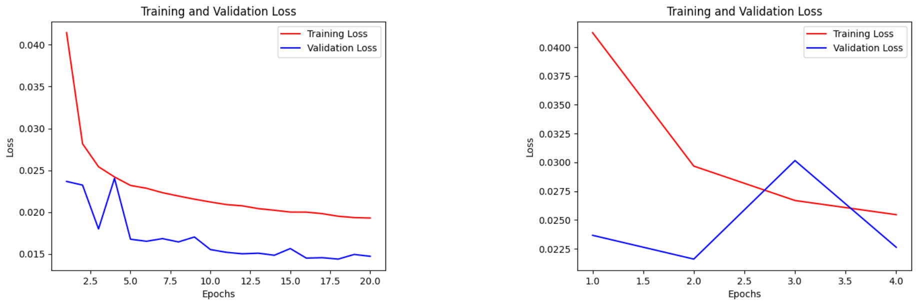


Figure 5.5: MLP Model 4 - CM



(a) MLP Model 4 - Best Fold (b) MLP Model 4 - Worst Fold

Figure 5.6: MLP Model 4 - Best and Worst Folds

Table 5.10: Best Models

Model	**AUC**	**F1**	**Precision**	**Recall**	**Accuracy**
Random Forest	99.99	99.66	99.66	99.67	99.67
XGBoost	99.99	99.65	99.65	99.66	99.66
MLP	99.94	99.42	99.44	99.46	99.46

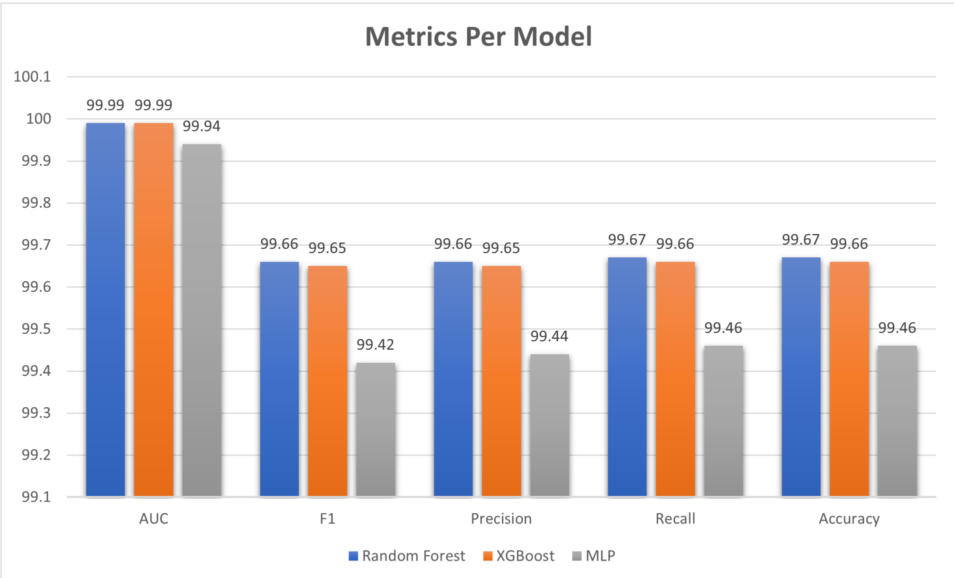


Figure 5.7: Models Grouped By Metric

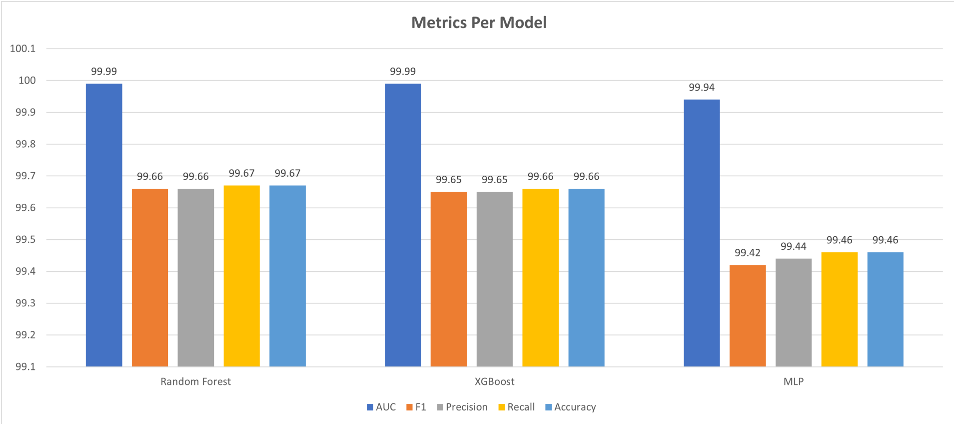


Figure 5.8: Metrics Grouped By Model

5.5 Feature Importance

Random Forest and Extreme Gradient Boosting both have feature importances that can be used to interpret each model's reliance on each chosen feature. After modelling, these were examined, and the following observations were made:

- Within the Random Forest Classifier, the top 4 features of importance were a range of 802.11w, non-802.11w and application layer features, such as *http.request.method, ip.ttl, radiotap.dbm_antsignal.*
- Within the XGBoost Classifer, top features include more non-802.11w and application layer features such as *ARP, ip.ttl, tcp.checksum.status, http.request.method, http.content_type, dns.*
- Of the application layer features, the HTTP request method was of significant importance; this is proposed to be due to SSDP's attack behaviour, which relies upon the 'M_Search' request type for a successful attack.
- *ip.ttl* (Time to live) was consistently ranked as an important feature in the context of XGBoost and Random Forest. The time to live is an IP feature that defines the maximum number of hops the packet travels before it stops.

5.6 Limitations

The experiments faced several limitations that should be considered in interpreting results and impact findings in this dissertation. Firstly, the constraints of time and computational resources hindered the ability to construct and test an exhaustive list of models, and the frequent occurrence of hardware and OS crashes negatively disrupted model training, leading to a loss in progress. In reference to the feature importance, this was explored briefly and was not fully used for feature selection or additional optimisation.

Lastly, the iterative and exploratory approach taken for model selection and tuning could be a source of error. This may have been susceptible to undue bias or randomness, potentially leading to under-explored models. These limitations prove that although the results are high, consideration should be exercised with caution with evaluating their use for a proposed IDS. Additional research is needed to validate these models, and further work is suggested to investigate other, more complex and tuned models.

5.7 Recommendations

Comparing all three machine learning algorithms (see Table 5.10 and Figures 5.7, 5.8), experiments from this study show that the Random Forest model ran with default parameters displayed slightly better results than the other two. The XGBoost model also performed very well with similar results but had a slight increase in false positives for Botnet, Malware and SSH. The MLP model, whilst still displaying high-performance levels, appeared to struggle with classifying minority classes. The confusion matrix contains more false negatives for Botnet, Malware and SSH, suggesting it could not predict these classes accurately.

As mentioned, identifying the most optimal solution for a proposed Wireless Network Intrusion Detection System is a highly subjective challenge. The final decision ultimately depends on the performance of the models in the environment and the particular needs and focuses in the production environment. The code for each high-performing model will be in the Appendix G and the project's code base.

6 Conclusion

6.1 Summary Of Findings

This project proposed the idea of utilising a set of application layer features together with 802.11 and non-802.11 features to develop machine learning models capable of distinguishing between six different attacks (Botnet, Malware, SQL Injection, SSDP Amplification, SSH and Website Spoofing) launched from the application layer from the AWID3 dataset. In total, 13 application layer features, detailed in Table 3.1, were chosen to help form the feature set used in each model. Two classifiers: Random Forest and XGBoost, and one deep neural network: Multilayer Perceptron machine learning algorithms, were used to evaluate their effectiveness on the problem.

Random Forest showed exceptional performance with default parameters; the AUC and F1 scores on the test set were incredibly high at 99.99% and 99.65%, providing a great balance of precision and recall. It was able to almost completely distinguish the SSDP class from the others. Naturally, due to the dataset imbalance, it struggled with misclassifications of minority classes, particularly Botnet, Malware, SQL and SSH, with recall values ranging from 0.77-0.82.

XGBoost, an ensemble method based on gradient boosting, carried similar results to RandomForest. Metrics of the highest achieving model on the test set shared similar performances of 99.99% AUC and 99.65% F1. In particular, it achieved perfect classification on the SSDP class, with zero misclassifications and the highest recall on the SQL Injection class. Like RF, it faced difficulty in detecting Botnet, Malware and SSH.

The MLP models showed a good overall performance but were significantly behind the other two models with AUC of 99.94% and 99.42% in F1. The highest-performing model consisted of 3 hidden layers with dropout, batch normalisation and ReLU as the activator. The AUC, F1, Recall and Precision were high but comparatively lower than XGBoost and RandomForest models. Like the two other models, it also struggled with under-represented classes. However, the MLP model struggled to identify the SQL Injection class, with a misclassification rate of 63% and a recall of 0.37.

Overall, this project shows that implementing application layer features with 802.11 and non-802.11 features achieved impressive performance with AUC, F1, Precision, Recall and Accuracy all above 99% in all chosen ML algorithms.

6.2 Project Review

The project deviated slightly from the original concept of detecting wireless network attacks on 802.11 networks; after a review of the existing literature, a gap was identified in exploring if machine learning could be leveraged to detect attacks launched from higher-level layers such as the Application Layer instead. The project diverted from its original timeline, and the struggle with manipulating the large dataset caused issues and hindered the time allocated to model training and evaluation. Additionally, the time required to tune parameters and create and evaluate models took much longer than initially thought, and the time allotted for this needed to be more. In light of this, a significant amount of helpful research and work was achieved, and the core objectives of this project were met.

6.3 Objectives

This project set out with the aim of meeting a series of objectives:

- 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, proving a recommendation for a proposed Wireless Intrusion Detection System (WIDS)

Objective: Existing Literature & Common ML Algorithms

Through the research, a plethora of existing research and literature were reviewed and considered for this project. Papers on wireless network standards, security attacks, intrusion detection systems and machine learning algorithms were reviewed extensively. The literature review section revealed a gap in research in exploring the detection of application layer attacks on wireless 802.11 networks, explicitly utilising the AWID3 dataset. Therefore, the exploration and examination of current literature on using ML for intrusion detection systems were met, including identifying common ML algorithms for classifying network attacks.

Objective: Training ML Models

Models were trained and tested using supervised machine learning techniques, specifically Random Forest, XGBoost and Multi-layer Perceptron. For most machine learning algorithms, a systematic approach was followed by establishing a baseline model with default parameters. Then, parameters and settings were tuned and changed to optimise the performance.

Each model subsequently achieved high-performance levels, with metrics such as AUC and F1 achieving scores of up to 99.9%, indicating a solid ability to classify the six attack classes launched from the application layer on an 802.11 wireless network dataset and thus achieve the objective.

Objective: Model Performance

Finally, in the analysis section, the performance of each model was analysed and compared using multiple metrics such as Classification Reports and Confusion Matrices and ranked. The best-performing models were then introduced and recommended as viable options to be implemented in a Wireless Network Intrusion Detection System (WIDS), meeting the final objective.

6.4 Contributions

This project focused on classifying six application layer attacks on an 802.11w wireless network dataset, specifically AWID3. This research builds upon similar work but combines a new selection of application layer features chosen from the 254 total features alongside 802.11 and non-802.11 features from (Chatzoglou, Kambourakis, Kolias, and Smiliotopoulos 2022a). Then, three machine learning algorithms: Random Forest, XGBoost and Multi-Layer Perceptron, were compared and evaluated; the best-performing models achieved high metrics of up to 99.9% in Area Under Curve (AUC) and 99.66% in F1 Score. The findings are unique and original and contribute to the existing literature and research surrounding machine learning in the context of Wireless Network Intrusion Detection Systems.

6.5 Limitations

As with all research, this project experience limitations and struggles that must be considered during interpretation. Due to the scope of the research, the hardware and time constraints impacted the complexity and range of exploration in the number of models, algorithms and parameters to be developed and tested. As such, hyperparameter tuning was limited, and the use the grid searching techniques such as GridSearchCV and RandomisedSearchCV was limited, resorting to a more iterative and exploratory method, potentially limiting the optimal parameters for each model. The feature importance of XGBoost and RF was explored to gain an understanding of the model. However, the exploration could have been more exhaustive and directly used for model optimisation.

A limitation of this research was the reliance on a single dataset; whilst the AWID3 dataset is well-established and used, the research and models may differ from real-world traffic and datasets with different data distributions.

In classification problems, imbalanced datasets can lead to biased models; no data balancing techniques, such as Under/Oversampling, were used for this research. Additionally, the focus was placed on achieving a balance between specificity and sensitivity, both critical factors in an Intrusion Detection System; therefore, other interpretations may draw different conclusions in model selection.

Finally, this research did focus on metrics such as training time, as this can be a crucial factor to consider when deployed in the real world.

6.6 Future Work

Additional work can explore the realms of including additional or different application layer features, considering each model's feature importance to determine their impacts on performance. Additionally, future work may address the limitations encountered during this research, using more advanced deep neural networks such as Convolutional Neural Networks (CNNs) or Stacked AutoEncoders and utilising larger parameter grids for hyper-parameter tuning. Moreover, using data balancing techniques and additional feature selection and engineering could be used to improve the performances and results of this project.

Finally, using the models in the real world would be a major milestone for future work as existing literature and research primarily focus on using datasets rather than in a real-world network environment. However, the contributions of this project will allow for a plethora of future work to be conducted.

References

A. Reyes, Abel et al. (2020). "A Machine Learning Based Two-Stage Wi-Fi Network Intrusion Detection System". In: *Electronics* 9.10. ISSN: 2079-9292. DOI: 10.3390/electronics9101689. URL: https://www.mdpi.com/2079-9292/9/10/1689.

Agrawal, Anand, Urbi Chatterjee, and Rajib Ranjan Maiti (2022). "KTRACKER: Passively Tracking KRACK Using ML Model". In: *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*. CODASPY '22. Baltimore, MD, USA: Association for Computing Machinery, pp. 364–366. ISBN: 9781450392204. DOI: 10.1145/3508398.3519360. URL: https://doi.org/10.1145/3508398.3519360.

Brownlee, Jason (July 2018). *Difference Between a Batch and an Epoch in a Neural Network*. Machine Learning Mastery. URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.

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, E., G. Kambourakis, C. Kolias, and C. Smiliotopoulos (2022a). "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.

— (2022b). "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.

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.

Dhanya, K.A. et al. (2023). "Detection of Network Attacks using Machine Learning and Deep Learning Models". In: *Procedia Computer Science* 218. International Conference on Machine Learning and Data Engineering, pp. 57–66. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2022.12.401. URL: https://www.sciencedirect.com/science/article/pii/S1877050922024942.

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).

Ge, Mengmeng et al. (2019). "Deep Learning-Based Intrusion Detection for IoT Networks". In: *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 256–25609.

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.

Kachavimath, Amit V, Shubhangeni Vijay Nazare, and Sheetal S Akki (2020). "Distributed Denial of Service Attack Detection using Naïve Bayes and K-Nearest Neighbor for Network Forensics". In: *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pp. 711–717. DOI: 10.1109/ICIMIA48430.2020.9074929.

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, C. 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.

Le, Thi-Thu-Huong, Yustus Eko Oktian, and Howon Kim (2022). "XGBoost for Imbalanced Multiclass Classification-Based Industrial Internet of Things Intrusion Detection Systems". In: *Sustainability* 14.14. ISSN: 2071-1050. DOI: 10.3390/su14148707. URL: https://www.mdpi.com/2071-1050/14/14/8707.

Liu, Gaoyuan et al. (2022). "An Enhanced Intrusion Detection Model Based on Improved kNN in WSNs". In: *Sensors* 22.4. ISSN: 1424-8220. DOI: 10.3390/s22041407. URL: https://www.mdpi.com/1424-8220/22/4/1407.

Lopez Perez, Rocio et al. (2018). "Machine Learning for Reliable Network Attack Detection in SCADA Systems". In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science*

And Engineering (TrustCom/BigDataSE), pp. 633–638. DOI: 10.1109/TrustCom/BigDataSE.2018.00094.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Moustafa, Nour and Jill Slay (2015). "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

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.

Ran, Jing, Yidong Ji, and Bihua Tang (2019). "A Semi-Supervised Learning Approach to IEEE 802.11 Network Anomaly Detection". In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pp. 1–5. DOI: 10.1109/VTCSpring.2019.8746576.

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).

Saini, Rahul, Debajyoti Halder, and Anand M. Baswade (2022). "RIDS: Real-time Intrusion Detection System for WPA3 enabled Enterprise Networks". In: *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*. http://arxiv.org/pdf/2207.02489, pp. 43–48. DOI: 10.1109/globecom48099.2022.10001501. URL: https://app.dimensions.ai/details/publication/pub.1154431160.

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.

Siriwardhana, Yushan (Dec. 2022). *5G-NIDD: A Comprehensive Network Intrusion Detection Dataset Generated over 5G Wireless Network*. https://doi.org/10.23729/e80ac9df-d9fb-47e7-8d0d-01384a415361. Yushan Siriwardhana.

Sivalingam, Krishna M. (2021). "Applications of Artificial Intelligence, Machine Learning and related techniques for Computer Networking Systems". In: *CoRR* abs/2105.15103. arXiv: 2105.15103. URL: https://arxiv.org/abs/2105.15103.

Smys, S, Abul Basar, Haoxiang Wang, et al. (2020). "Hybrid intrusion detection system for internet of things (IoT)". In: *Journal of ISMAC* 2.04, pp. 190–199.

Torrres, Anibal (2021). *WiFi Anomaly Behavior Analysis Based Intrusion Detection Using Online Learning*. URL: http://hdl.handle.net/10150/666297.

Uszko, Krzysztof et al. (2023). "Rule-Based System with Machine Learning Support for Detecting Anomalies in 5G WLANs". In: *Electronics* 12.11. ISSN: 2079-9292. DOI: 10.3390/electronics12112355. URL: https://www.mdpi.com/2079-9292/12/11/2355.

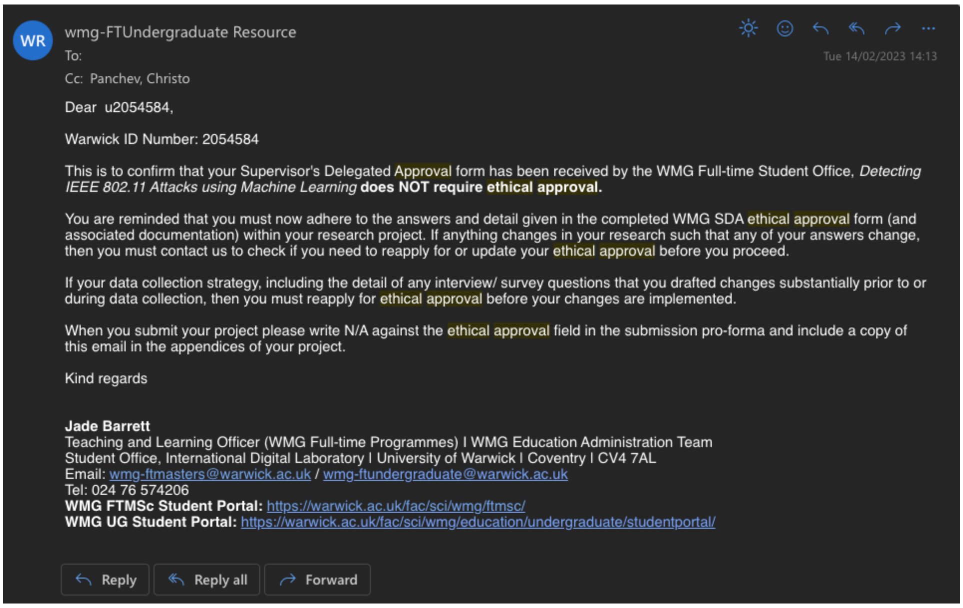
Vanhoef, Mathy and Frank Piessens (2017). "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, pp. 1313–1328. ISBN: 9781450349468. DOI: 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).

Appendices

A Ethical Approval



WR
wmg-FTUndergraduate Resource
To:
Cc: Panchev, Christo
Tue 14/02/2023 14:13

Dear u2054584,

Warwick ID Number: 2054584

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.**

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.

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.

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.

Kind regards

Jade Barrett
Teaching and Learning Officer (WMG Full-time Programmes) I WMG Education Administration Team
Student Office, International Digital Laboratory I University of Warwick I Coventry I CV4 7AL
Email: wmg-ftmasters@warwick.ac.uk / wmg-ftundergraduate@warwick.ac.uk
Tel: 024 76 574206
WMG FTMSc Student Portal: https://warwick.ac.uk/fac/sci/wmg/ftmsc/
WMG UG Student Portal: https://warwick.ac.uk/fac/sci/wmg/education/undergraduate/studentportal/

Reply | Reply all | Forward

```
Dear 2054584,

Warwick ID Number: 2054584

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.

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.

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.

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.
```

```

Kind regards

Jade Barrett
```

B Dataset Manipulation

B.1 CSV Combiner Script

```bash
#!/bin/bash

# Input Directory
input_dir="../Datasets"

cd "$input_dir"

# Set the output file name
output_file="../Datasets/combined/combined.csv"

# Check if the output file already exists and delete it
if [ -f "$output_file" ]; then
  rm "$output_file"
fi

echo "Combining files..."

# Loop through all the files
for file in $(ls *_reduced.csv | sort -V)
do
  # Check if the file exists
  if [ -f "$file" ]; then

    echo "Combining $file..."

    # For the first file, copy the header to the output file
    if [ ! -f "$output_file" ]; then
      head -n 1 "$file" > "$output_file"
    fi

    # Append all the rows except the header to the output file
    tail -n +2 "$file" >> "$output_file"
  fi
done

echo "Combining Finished"
```

B.2 Feature Extraction & Reduction

```
# Define the columns to extract
cols_to_use = [
    'frame.len','radiotap.dbm_antsignal','radiotap.length',
    'wlan.duration','wlan_radio.duration','wlan_radio.signal_dbm',
    'radiotap.present.tsft','wlan.fc.type','wlan.fc.subtype',
    'wlan.fc.ds','wlan.fc.frag','wlan.fc.moredata',
    'wlan.fc.protected','wlan.fc.pwrmgt','wlan.fc.retry',
    'wlan_radio.phy','udp.length','ip.ttl',
    'arp','arp.proto.type','arp.hw.size',
    'arp.proto.size','arp.hw.type','arp.opcode',
    'tcp.analysis','tcp.analysis.retransmission','tcp.option_len',
    'tcp.checksum.status','tcp.flags.ack','tcp.flags.fin',
    'tcp.flags.push','tcp.flags.reset','tcp.flags.syn',
    'dns','dns.count.queries','dns.count.answers',
    'dns.resp.len','dns.resp.ttl','http.request.method',
    'http.response.code','http.content_type','ssh.message_code',
    'ssh.packet_length','nbns','nbss.length',
    'nbss.type','ldap','smb2.cmd',
    'smb.flags.response','smb.access.generic_read',
    'smb.access.generic_write','smb.access.generic_execute',
    'Label']

batch_size = 1000000

combined_df = pd.DataFrame()

# Iterate through the file in batches
for chunk in pd.read_csv('botnet_combined.csv', chunksize=
        batch_size, usecols=cols_to_use, low_memory=False):

    # Combine the processed chunk with previous chunks
    combined_df = pd.concat([combined_df, chunk])
```

```
# Drop all missing rows that contain only nan values
combined_df = combined_df.dropna(how='all')

# Drop all rows with missing values in Label Column
combined_df = combined_df.dropna(subset=['Label'])

# Fill NAs with zeros
# Change nan values to 0
combined_df = combined_df.fillna(0)
```

```
# Duplicate the dataframe
df = combined_df.copy()

# Regex to keep only the first value e.g
# -100-100-10 becomes -100,  123-456-1 becomes 123, -10-2
    becomes-10, 81-63-63 becomes 81
def seperated_values(x):
    x = str(x)
    match = re.match(r'^(-?\d+).*$', x)
    if match:
        return match.group(1)
    else:
        return x

# Go through all columns and change seperate values into just
    one value
for column in df.columns:
    df[column] = df[column].apply(seperated_values)
    print('Processing', column)
print('Done')

# 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.
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"

# Use str.match method to apply the regex pattern to the column
mask = df['tcp.option_len'].astype(str).str.match(regex).fillna(
    False)
df = df[~mask]

mask = df['dns.resp.ttl'].astype(str).str.match(regex).fillna(
    False)
df = df[~mask]

mask = df['ip.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask]

mask = df['smb2.cmd'].astype(str).str.match(regex).fillna(False)
df = df[~mask]

df.to_csv('Botnet_Reduced.csv', index=False)
```

C Conda Environments

C.1 Neural Networks - Apple Silicon

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

C.2 Classifiers

```
# Conda environment used for Random Forest, XGBoost and K-NN.

conda create -n ml-env python=3.9
conda activate ml-env
conda install -c conda-forge -y pandas jupyter
pip install numpy, matplotlib, scikit-learn, scipy, seaborn, xgboost
```

D Data Preprocessing

D.1 MinMax Scaling

```
# Define the scaler
scaler = MinMaxScaler()

# Fit the scaler to the following columns we define
scale_cols = [
        'frame.len',
        'radiotap.dbm_antsignal',
        'radiotap.length',
        'wlan.duration',
        'wlan_radio.duration',
        'wlan_radio.signal_dbm',
        'ip.ttl',
        'udp.length',
        'nbss.length',
        'dns.count.answers',
        'dns.count.queries',
        'dns.resp.ttl',
        'ssh.packet_length']

# Fit the X_train and X_test
X_train[scale_cols] = scaler.fit_transform(X_train[scale_cols])
X_test[scale_cols] = scaler.transform(X_test[scale_cols])
```

D.2 OHE Encoding

```
cols_to_encode = [col for col in X_train.columns if col not in
    scale_cols]
X_all = pd.concat([X_train, X_test], axis=0)

X_all_ohe = pd.get_dummies(X_all, columns=cols_to_encode,
    drop_first=True, dtype=np.uint8)

# split back into train and test sets
X_train_ohe = X_all_ohe[:len(X_train)]
X_test_ohe = X_all_ohe[len(X_train):]
```

D.3 Label Encoding

```
    # Use Label Encoder to encode the target variable
le = LabelEncoder()

label_encoder = le.fit(y_train)
y_train_encoded = label_encoder.transform(y_train)
```

D.4 Loading Dataset

```
chunk_size = 1000000
dtype_opt = {
    'frame.len': 'int64',
    'radiotap.dbm_antsignal': 'int64',
    'radiotap.length': 'int64',
    'radiotap.present.tsft': 'int64',
    'wlan.duration': 'int64',
    'wlan.fc.ds': 'int64',
    'wlan.fc.frag': 'int64',
    'wlan.fc.moredata': 'int64',
    'wlan.fc.protected': 'int64',
    'wlan.fc.pwrmgt': 'int64',
    'wlan.fc.type': 'int64',
    'wlan.fc.retry': 'int64',
    'wlan.fc.subtype': 'int64',
    'wlan_radio.duration': 'int64',
    'wlan_radio.signal_dbm': 'int64',
    'wlan_radio.phy': 'int64',
    'arp': 'object',
    'arp.hw.type': 'object',
    'arp.proto.type': 'int64',
    'arp.hw.size': 'int64',
    'arp.proto.size': 'int64',
    'arp.opcode': 'int64',
    'ip.ttl': 'int64',
    'tcp.analysis': 'int64',
    'tcp.analysis.retransmission': 'int64',
    'tcp.checksum.status': 'int64',
    'tcp.flags.syn': 'int64',
    'tcp.flags.ack': 'int64',
    'tcp.flags.fin': 'int64',
    'tcp.flags.push': 'int64',
    'tcp.flags.reset': 'int64',
    'tcp.option_len': 'int64',
    'udp.length': 'int64',
    'nbns': 'object',
    'nbss.length': 'int64',
    'ldap': 'object',
    'smb2.cmd': 'int64',
```

```
    'dns': 'object',
    'dns.count.answers': 'int64',
    'dns.count.queries': 'int64',
    'dns.resp.ttl': 'int64',
    'http.content_type': 'object',
    'http.request.method': 'object',
    'http.response.code': 'int64',
    'ssh.message_code': 'int64',
    'ssh.packet_length': 'int64'
}

# Read the data
print('Reading X...')
X = pd.DataFrame()
for chunk in pd.read_csv('X.csv', chunksize=chunk_size, usecols=
    dtype_opt.keys(), dtype=dtype_opt, low_memory=False):
    X = pd.concat([X, chunk])

print('Reading y...')
y = pd.DataFrame()
for chunk in pd.read_csv('y.csv', chunksize=chunk_size, usecols
    =['Label'], dtype='object', low_memory=False):
   y = pd.concat([y, chunk])

# Split the data into training and testing sets
print('Splitting the data...')
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)

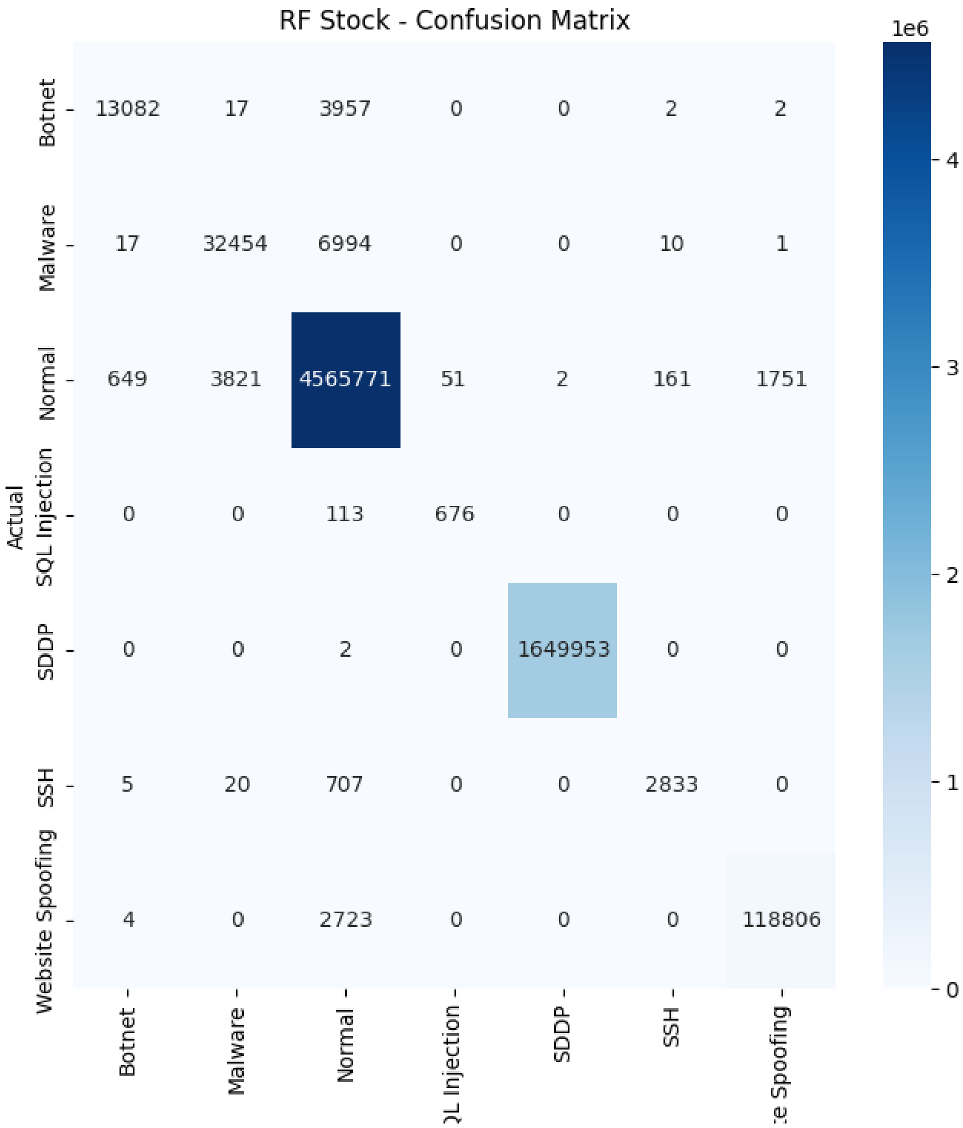
```
# Use KNN
from sklearn.neighbors import KNeighborsClassifier

k=5

# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors=k, n_jobs=-1)

# Fit the model
knn.fit(X_train_ohe, y_train_encoded)

# predict the test set
y_knn_pred = knn.predict(X_test_ohe)

from sklearn.metrics import classification_report, roc_auc_score

# Get the classification report
report = classification_report(y_test_encoded, y_knn_pred)

print('Classification Report:\n', report)

# Get the all the metrics for the multi class classification

print('Accuracy: ', accuracy_score(y_test_encoded, y_knn_pred))
print('Precision: ', precision_score(y_test_encoded, y_knn_pred,
      average='macro'))
print('Recall: ', recall_score(y_test_encoded, y_knn_pred,
      average='macro'))
print('F1 Score: ', f1_score(y_test_encoded, y_knn_pred, average
      ='macro'))

# Get the confusion matrix for multi-class and plot it
confusion = confusion_matrix(y_test, y_rf_pred)
print('Confusion Matrix\n')
print(confusion)

# Plot the confusion matrix for multi-class classification using
      seaborn
labels = ['Normal', 'SSDP', 'Website Spoofing', 'Malware', '
      Botnet', 'SSH', 'SQL Injection']

plt.figure(figsize=(8, 8))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',
      xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

```

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

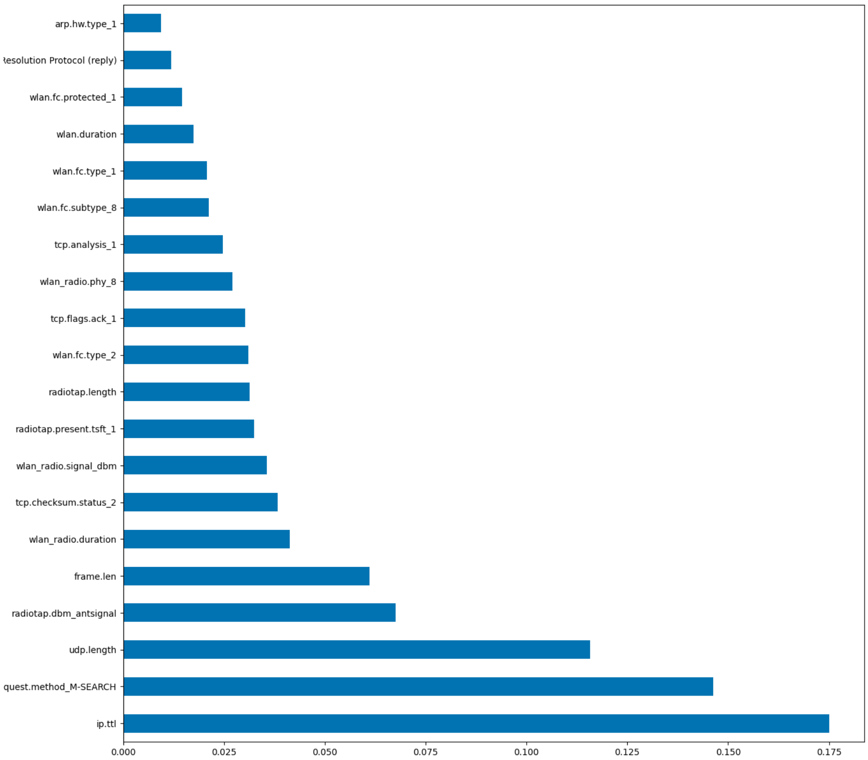
E.2 Random Forest

E.2.1 RF Model ID 0 - Raw Metrics

```
S-CV Results
Mean AUC = 99.99
Mean F1 = 99.66
Mean Precision = 99.66
Mean Recall = 99.67
Mean Accuracy = 99.67
Training Time: 7795 seconds

Final Test Results
Test AUC: 0.9999070506312879
Weighted Test F1: 0.996638797834701
Weighted Test Precision: 0.9966379719195173
Weighted Test Recall: 0.9967196932696956
Test Accuracy: 0.9967196932696956

Classification Report
                precision    recall  f1-score   support

      Botnet       0.95      0.77      0.85     17060
     Malware       0.89      0.82      0.86     39476
      Normal       1.00      1.00      1.00    457220
        SQL        0.93      0.86      0.89       789
        SSDP       1.00      1.00      1.00   1649955
        SSH        0.94      0.79      0.86      3565
    Spoofing       0.99      0.98      0.98    121533

    accuracy                           1.00   6404584
   macro avg       0.96      0.89      0.92   6404584
weighted avg       1.00      1.00      1.00   6404584

Confusion Matrix
[[  13082      17    3957       0       0       2       2]
 [     17   32454    6994       0       0      10       1]
 [    649    3821 4565771      51       2     161    1751]
 [      0       0     113     676       0       0       0]
 [      0       0       2       0 1649953       0       0]
 [      5      20     707       0       0    2833       0]
 [      4       0    2723       0       0       0  118806]]
```

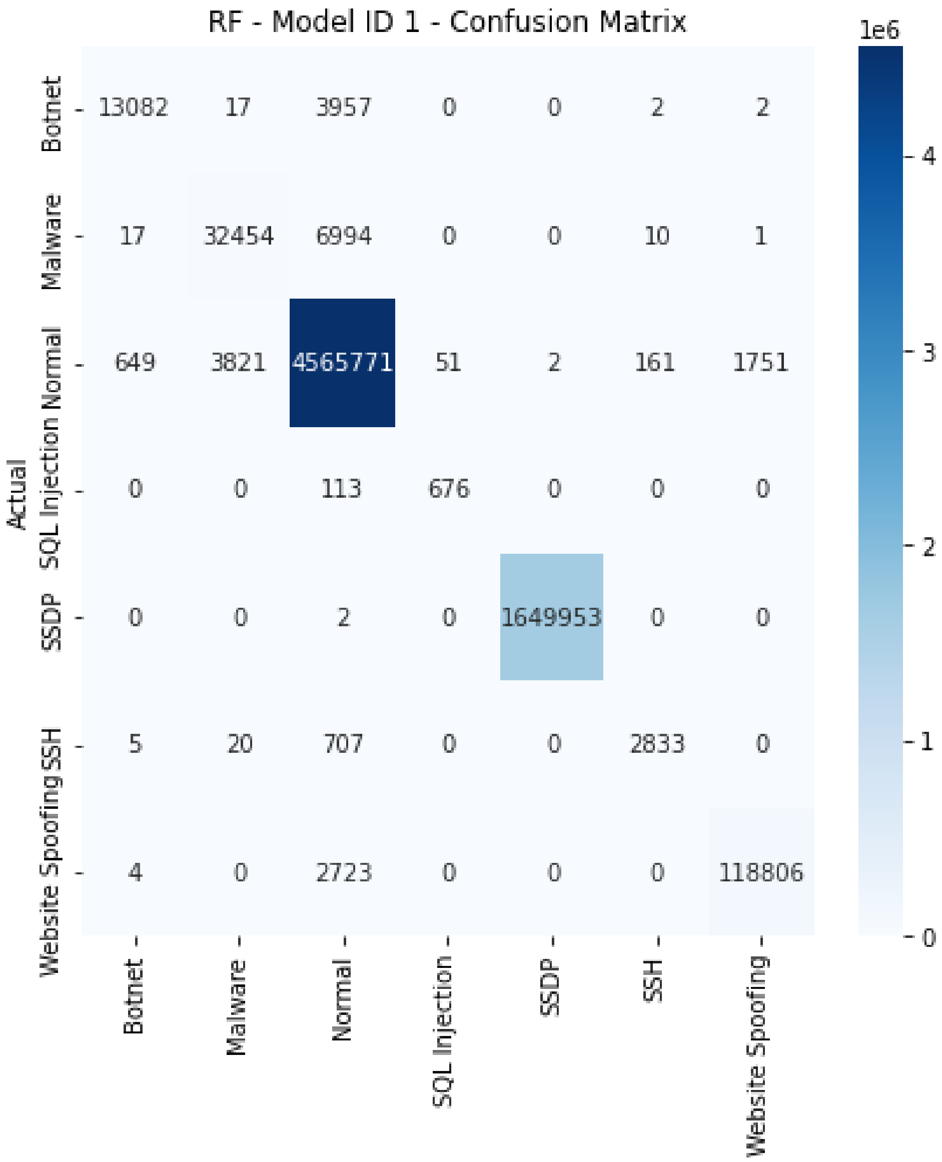


RF Model 0 - Confusion Matrix

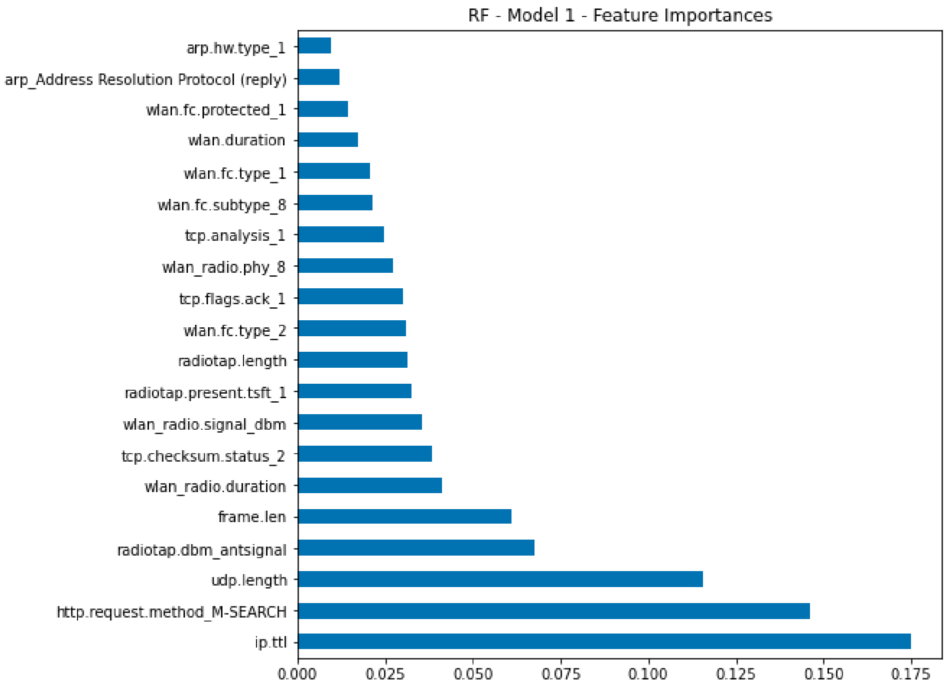

arp.hw.type_1
Resolution Protocol (reply)
wlan.fc.protected_1
wlan.duration
wlan.fc.type_1
wlan.fc.subtype_8
tcp.analysis_1
wlan_radio.phy_8
tcp.flags.ack_1
wlan.fc.type_2
radiotap.length
radiotap.present.tsft_1
wlan_radio.signal_dbm
tcp.checksum.status_2
wlan_radio.duration
frame.len
radiotap.dbm_antsignal
udp.length
quest.method_M-SEARCH
ip.ttl
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

E.2.2 RF Model ID 1 - Raw Metrics

```
S-CV Results
Mean AUC = 99.99
Mean F1 = 99.66
Mean Precision = 99.66
Mean Recall = 99.67
Mean Accuracy = 99.67
Training Time 7794.549654006958 seconds

Final Test Results
Test AUC: 0.9999070506312879
Weighted Test F1: 0.996638797834701
Weighted Test Precision: 0.9966379719195173
Weighted Test Recall: 0.9967196932696956
Test Accuracy: 0.9967196932696956

Classification Report
               precision    recall  f1-score   support

      Botnet        0.95      0.77      0.85     17060
     Malware        0.89      0.82      0.86     39476
      Normal        1.00      1.00      1.00   4572206
         SQL        0.93      0.86      0.89       789
        SSDP        1.00      1.00      1.00   1649955
         SSH        0.94      0.79      0.86      3565
WebsiteSpoof        0.99      0.98      0.98    121533

    accuracy                            1.00   6404584
   macro avg        0.96      0.89      0.92   6404584
weighted avg        1.00      1.00      1.00   6404584

Confusion Matrix
[[  13082      17    3957       0       0       2       2]
 [     17   32454    6994       0       0      10       1]
 [    649    3821 4565771      51       2     161    1751]
 [      0       0     113     676       0       0       0]
 [      0       0       2       0 1649953       0       0]
 [      5      20     707       0       0    2833       0]
 [      4       0    2723       0       0       0  118806]]
```



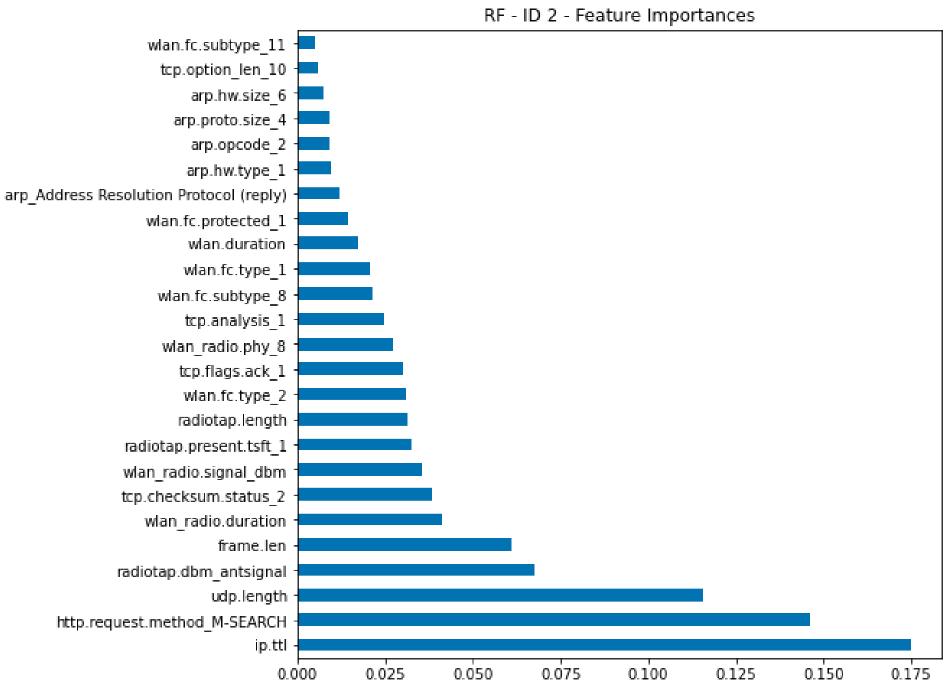
RF Model 1 - Confusion Matrix



RF Model 1 - Feature Importance

E.2.3 RF Model ID 2 - Raw Metrics

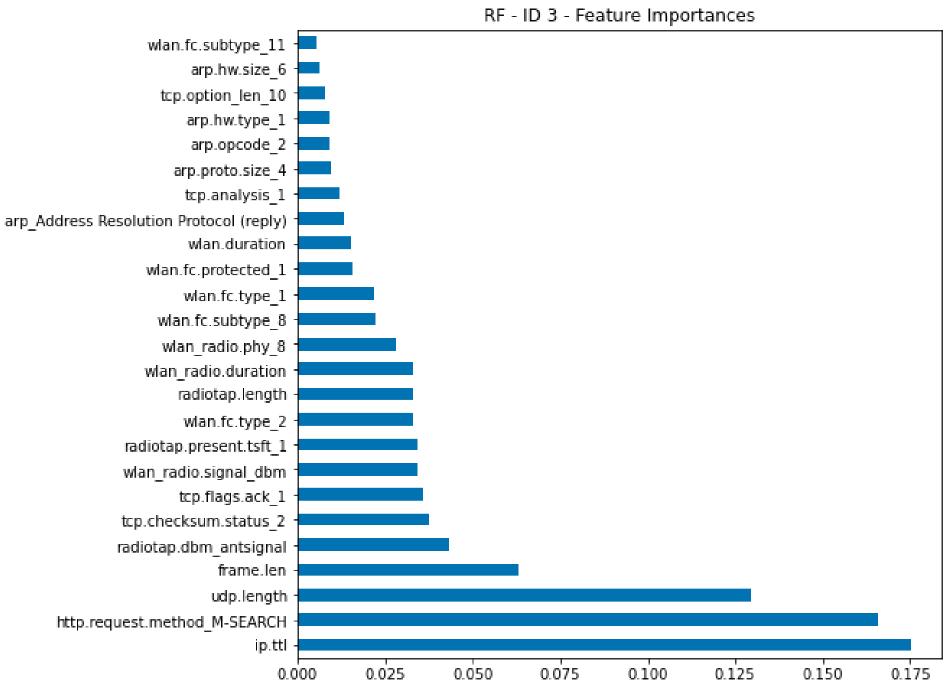
```
Final Test Results
Test AUC:   0.9999070506308619
Weighted Test Precision:   0.9966379719195173
Weighted Test Recall:   0.9967196932696956
Weighted Test F1:   0.996638797834701
Test Accuracy:   0.9967196932696956

Classification Report
                  precision    recall  f1-score   support

          Botnet       0.95      0.77      0.85     17060
         Malware       0.89      0.82      0.86     39476
          Normal       1.00      1.00      1.00   4572206
   SQL_Injection       0.93      0.86      0.89       789
            SSDP       1.00      1.00      1.00   1649955
             SSH       0.94      0.79      0.86      3565
Website_spoofing       0.99      0.98      0.98    121533

        accuracy                           1.00   6404584
       macro avg       0.96      0.89      0.92   6404584
    weighted avg       1.00      1.00      1.00   6404584

Confusion Matrix
[[  13082       17     3957        0        0        2        2]
 [     17    32454     6994        0        0       10        1]
 [    649     3821  4565771       51        2      161     1751]
 [      0        0      113      676        0        0        0]
 [      0        0        2        0  1649953        0        0]
 [      5       20      707        0        0     2833        0]
 [      4        0     2723        0        0        0   118806]]
```



RF Model 2 - Feature Importance

E.2.4 RF Model ID 3 - Raw Metrics

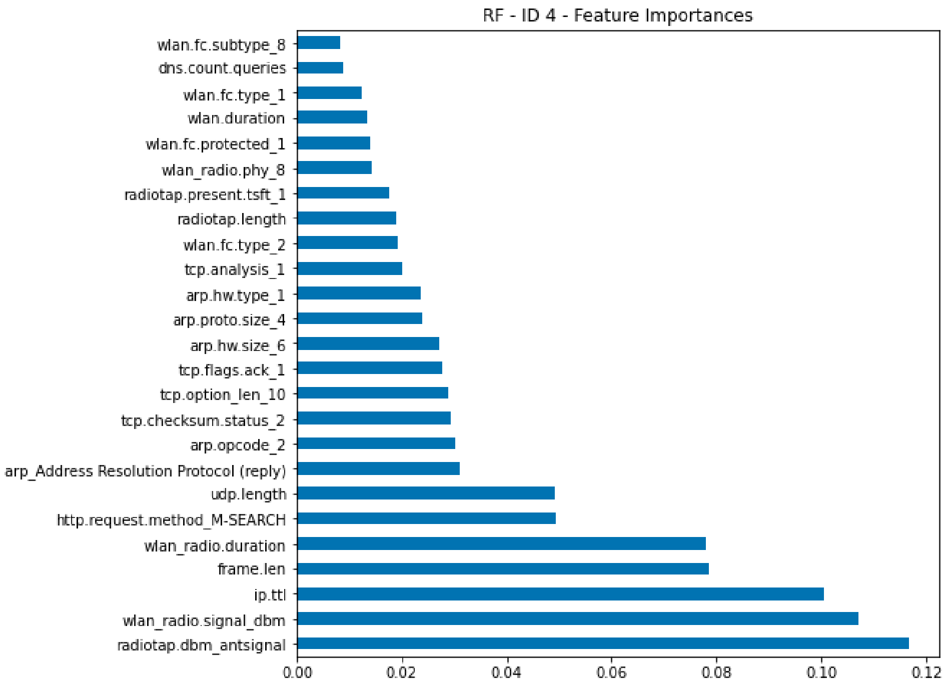
```
S-CV Results
Mean AUC = 0.9999
Mean F1 = 0.9966
Mean Precision = 0.9966
Mean Recall = 0.9967
Mean Accuracy = 0.9967
Training Time 56042.87267756462  seconds

Final Test Results
Test AUC:  0.9999070506308619
Weighted Test F1:  0.996638797834701
Weighted Test Precision:  0.9966379719195173
Weighted Test Recall:  0.9967196932696956
Test Accuracy:  0.9967196932696956

Classification Report
                  precision    recall  f1-score   support

          Botnet       0.95      0.77      0.85     17060
         Malware       0.89      0.82      0.86     39476
          Normal       1.00      1.00      1.00   4572206
   SQL_Injection       0.93      0.86      0.89       789
            SSDP       1.00      1.00      1.00   1649955
             SSH       0.94      0.79      0.86      3565
Website_spoofing       0.99      0.98      0.98    121533

        accuracy                           1.00   6404584
       macro avg       0.96      0.89      0.92   6404584
    weighted avg       1.00      1.00      1.00   6404584


Confusion Matrix

[[  13082      17    3957       0       0       2       2]
 [     17   32454    6994       0       0      10       1]
 [    649    3821 4565771      51       2     161    1751]
 [      0       0     113     676       0       0       0]
 [      0       0       2       0 1649953       0       0]
 [      5      20     707       0       0    2833       0]
 [      4       0    2723       0       0       0  118806]]
```



RF Model 3 - Feature Importance

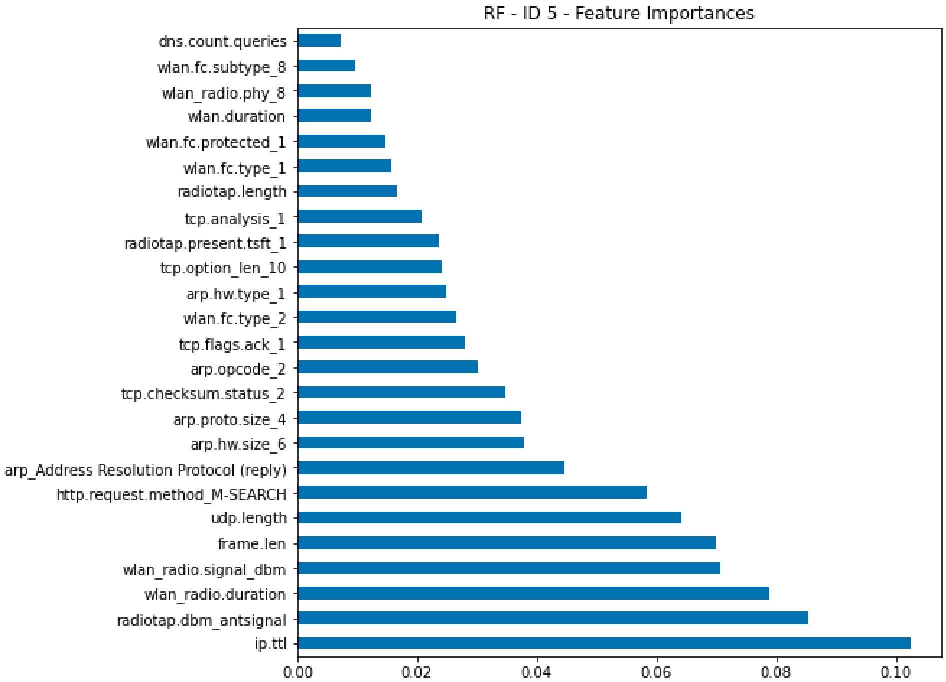
E.2.5 RF Model ID 4 - Raw Metrics

```
S-CV Results
Mean AUC = 99.95
Mean F1 = 95.23
Mean Precision = 98.50
Mean Recall = 92.96
Mean Accuracy = 92.96
Training Time = 10147 seconds

Final Test Results
Test AUC: 0.9994792868436975
Weighted Test Precision: 0.984757273176817
Weighted Test Recall: 0.925409987596384
Weighted Test F1: 0.9496738038716113
Test Accuracy: 0.925409987596384

Classification Report

                   precision    recall  f1-score   support

          Botnet       0.14      0.96      0.24     17060
         Malware       0.22      0.97      0.36     39476
          Normal       1.00      0.90      0.95   4572206
   SQL_Injection       0.01      0.99      0.02       789
            SSDP       1.00      1.00      1.00   1649955
             SSH       0.04      0.99      0.08      3565
Website_spoofing       0.61      0.98      0.75    121533

        accuracy                           0.93   6404584
       macro avg       0.43      0.97      0.48   6404584
    weighted avg       0.98      0.93      0.95   6404584


Confusion Matrix

[[  16326      90      30      91       0     504      19]
 [     76   38392      13     170       0     824       1]
 [ 102163  135709 4098890   76381       2   83351   75710]
 [      0       0       1     785       0       3       0]
 [      0       0      10       0 1649945       0       0]
 [      6      15       4      16       0    3523       1]
 [    282    1145     484     262       0     355  119005]]
```


RF - ID 4 - Feature Importances
wlan.fc.subtype_8
dns.count.queries
wlan.fc.type_1
wlan.duration
wlan.fc.protected_1
wlan_radio.phy_8
radiotap.present.tsft_1
radiotap.length
wlan.fc.type_2
tcp.analysis_1
arp.hw.type_1
arp.proto.size_4
arp.hw.size_6
tcp.flags.ack_1
tcp.option_len_10
tcp.checksum.status_2
arp.opcode_2
arp_Address Resolution Protocol (reply)
udp.length
http.request.method_M-SEARCH
wlan_radio.duration
frame.len
ip.ttl
wlan_radio.signal_dbm
radiotap.dbm_antsignal
0.00
0.02
0.04
0.06
0.08
0.10
0.12

E.2.6 RF Model ID 5 - Raw Metrics

```
S-CV Results
Mean AUC = 0.9987
Mean F1 = 0.9153
Mean Precision = 0.9842
Mean Recall = 0.8665
Mean Accuracy = 0.8665
Training Time 4632.155310869217 seconds

Final Test Results
Test AUC: 0.998635554230961
Weighted Test Precision: 0.9848384599880898
Weighted Test Recall: 0.8600619493787575
Weighted Test F1: 0.9116563847511311
Test Accuracy: 0.8600619493787575

Classification Report

              precision    recall  f1-score   support

      Botnet       0.06      0.94      0.12     17060
     Malware       0.10      0.90      0.19     39476
      Normal       1.00      0.81      0.89   4572206
         SQL       0.01      0.99      0.01       789
        SSDP       0.99      1.00      1.00   1649955
         SSH       0.02      0.99      0.04      3565
    WebSpoof       0.76      0.92      0.83    121533

    accuracy                           0.86   6404584
   macro avg       0.42      0.94      0.44   6404584
weighted avg       0.98      0.86      0.91   6404584


Confusion Matrix

[[  16064     141      26     133       0     693       3]
 [     72   35584     121     125       0    3572       2]
 [ 240322  301805 3690025  138640   11696  154013   35705]
 [      0       0       0     783       0       5       1]
 [      0       0       9       0 1649946       0       0]
 [      6       1       0      12       0    3546       0]
 [   4970    1711     301    1392       0     768  112391]]
```

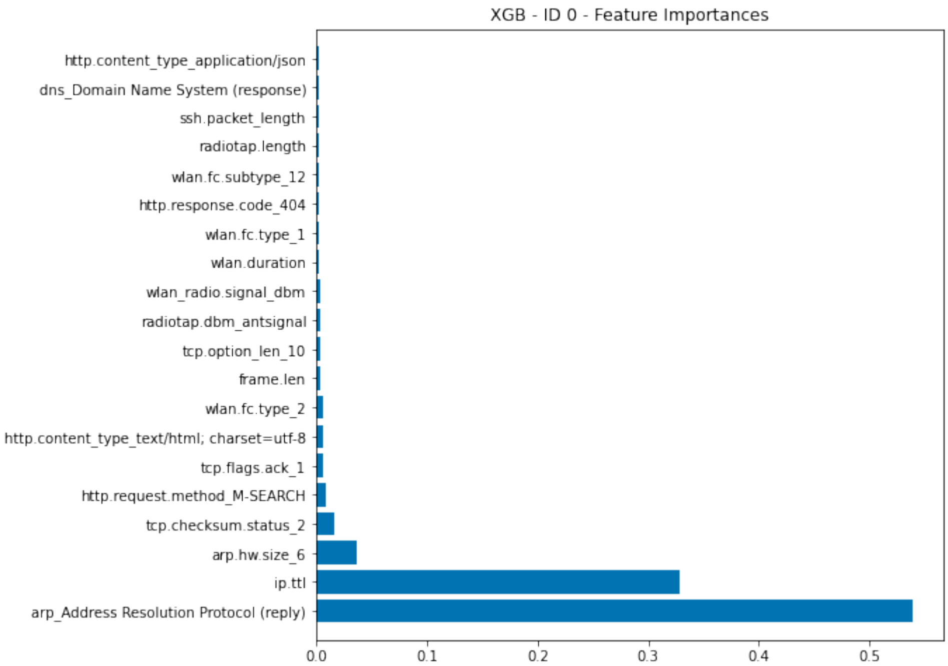


RF Model 5 - Feature Importance

E.3 XGBoost

E.3.1 XGBoost Model 0 - Raw Metrics

```

Final Test Results
Test AUC: 0.9999
Weighted F1: 0.9963
Weighted Precision: 0.9964
Weighted Recall: 0.9964
Test Accuracy: 0.9964

Classification Report
              precision    recall  f1-score   support

      Botnet       0.96      0.74      0.83     10236
     Malware       0.86      0.85      0.86     23686
      Normal       1.00      1.00      1.00   2743324
         SQL       0.95      0.87      0.91       473
        SSDP       1.00      1.00      1.00    989973
         SSH       0.90      0.77      0.83      2139
    WebSpoof       0.99      0.97      0.98     72920

    accuracy                           1.00   3842751
   macro avg       0.95      0.88      0.91   3842751
weighted avg       1.00      1.00      1.00   3842751

Confusion Matrix
[[   7538      29    2664       0       0       2       3]
 [     19   20191    3471       0       0       5       0]
 [    273    3282 2738552      20       1     185    1011]
 [      0       0      63     410       0       0       0]
 [      0       0       1       0  989972       0       0]
 [      4      11     484       0       0    1640       0]
 [      3       1    2319       0       0       0   70597]]
```

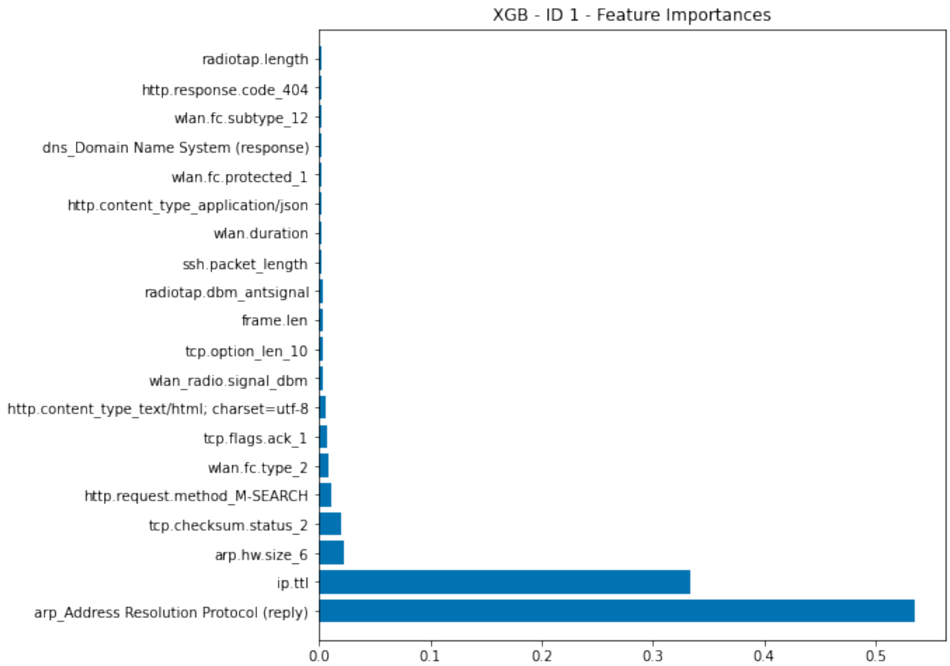


XGBoost Model 0 - Feature Importance

E.3.2 XGBoost Model 1 - Raw Metrics

```
S-CV Results


Final Test Results
Test AUC: 0.9999
Weighted Test F1: 0.9964
Weighted Test Precision: 0.9965
Weighted Test Recall: 0.9965
Test Accuracy: 0.9965

Classification Report
              precision    recall  f1-score   support

      Botnet       0.96      0.75      0.84     13648
     Malware       0.86      0.86      0.86     31581
      Normal       1.00      1.00      1.00   3657765
         SQL       0.94      0.84      0.89       631
        SSDP       1.00      1.00      1.00   1319964
         SSH       0.95      0.76      0.84      2852
    WebSpoof       0.99      0.97      0.98     97226

    accuracy                           1.00   5123667
   macro avg       0.96      0.88      0.91   5123667
weighted avg       1.00      1.00      1.00   5123667
```

```

Confusion Matrix
[[   10250      10    3382       0       0       1       5]
 [      40   27004    4533       0       0       3       1]
 [     406    4444 3651488      36       1     108    1282]
 [       0       0     101     530       0       0       0]
 [       0       0       6       0 1319958       0       0]
 [       3      19     670       0       0    2160       0]
 [       0       8    2841       0       0       0   94377]]
```

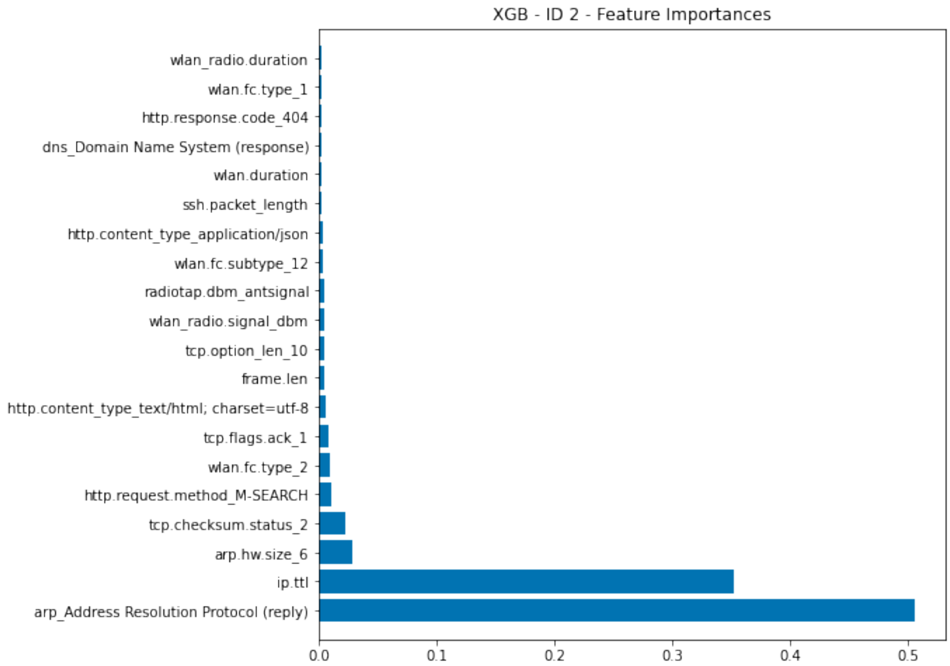


XGBoost Model 1 - Feature Importance

E.3.3 XGBoost Model 2 - Raw Metrics

```

Final Test Results
Test AUC: 0.9999
Weighted Test F1: 0.9964
Weighted Test Precision: 0.9964
Weighted Test Recall: 0.9964
Test Accuracy: 0.9964

Classification Report
              precision    recall  f1-score   support

```

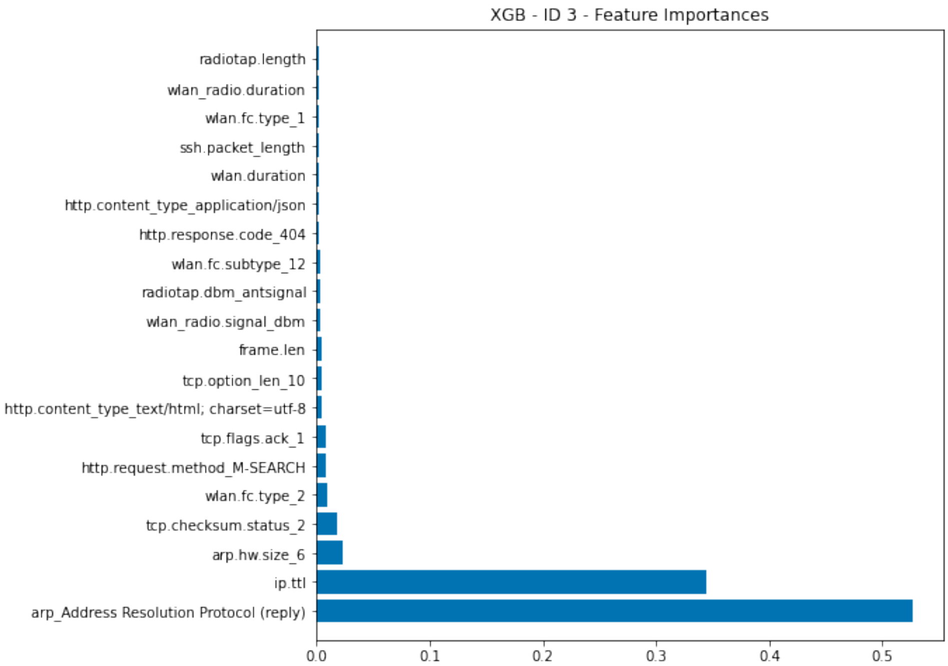
```
      Botnet       0.97      0.74      0.84     13648
     Malware       0.85      0.86      0.86     31581
      Normal       1.00      1.00      1.00   3657765
         SQL       0.93      0.84      0.88       631
        SSDP       1.00      1.00      1.00   1319964
         SSH       0.91      0.74      0.82      2852
    WebSpoof       0.99      0.97      0.98     97226

    accuracy                           1.00   5123667
   macro avg       0.95      0.88      0.91   5123667
weighted avg       1.00      1.00      1.00   5123667

Confusion Matrix
[[  10094      20    3528       0       0       2       4]
 [     34   27180    4365       0       0       2       0]
 [    331    4611 3651281      38       1     203    1300]
 [      0       0     103     528       0       0       0]
 [      0       0       6       0 1319958       0       0]
 [      1      19     714       0       0    2118       0]
 [      0       4    2950       0       0       0   94272]]
```



XGBoost Model 2 - Feature Importance

E.3.4 XGBoost Model 3 - Raw Metrics

```
Final Test Results
Test AUC: 0.9999
Weighted Test F1: 0.9964
Weighted Test Precision: 0.9964
Weighted Test Recall: 0.9964
Test Accuracy: 0.9964

Classification Report
               precision    recall  f1-score   support

      Botnet       0.96      0.74      0.84     13648
     Malware       0.85      0.86      0.86     31581
      Normal       1.00      1.00      1.00   3657765
         SQL       0.94      0.85      0.89       631
        SSDP       1.00      1.00      1.00   1319964
         SSH       0.92      0.74      0.82      2852
    WebSpoof       0.99      0.97      0.98     97226

    accuracy                           1.00   5123667
   macro avg       0.95      0.88      0.91   5123667
weighted avg       1.00      1.00      1.00   5123667

Confusion Matrix
[[  10133       9    3499       0       0       1       6]
 [     36   27171    4370       0       0       3       1]
 [    350    4621 3651253      35       1     193    1312]
 [      0       0      93     538       0       0       0]
 [      0       0       6       0 1319958       0       0]
 [      4      19     705       0       0    2124       0]
 [      0      10    2921       0       0       0   94295]]
```



XGBoost Model 3 - Feature Importance

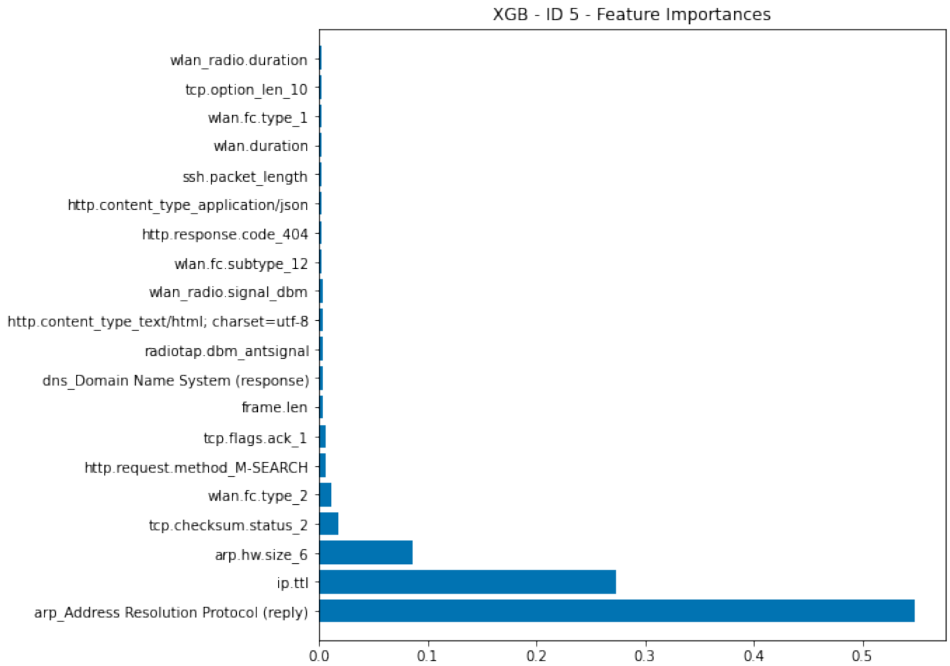
E.3.5 XGBoost Model 4 - Raw Metrics

```
GridSearchCV Results
Searching took 101767.660586 seconds
{'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 300}
0.9964800321988857
```

E.3.6 XGBoost Model 5 - Raw Metrics

```

Final Test Results
Test AUC: 0.9999
Weighted Test F1: 0.9964
Weighted Test Precision: 0.9965
Weighted Test Recall: 0.9965
Test Accuracy: 0.9965

Classification Report
              precision    recall  f1-score   support

      Botnet       0.96      0.75      0.84     13648
     Malware       0.85      0.86      0.86     31581
```

```
       Normal       1.00      1.00      1.00   3657765
          SQL       0.93      0.87      0.90       631
         SSDP       1.00      1.00      1.00   1319964
          SSH       0.91      0.76      0.83      2852
     WebSpoof       0.99      0.97      0.98     97226

     accuracy                           1.00   5123667
    macro avg       0.95      0.89      0.91   5123667
 weighted avg       1.00      1.00      1.00   5123667

Confusion Matrix
[[  10251      15    3378       0       0       2       2]
 [     35   27290    4239       0       0       5      12]
 [    424    4709 3651040      42       1     213    1336]
 [      0       0      84     547       0       0       0]
 [      0       0       6       0 1319958       0       0]
 [      0      19     671       0       0    2162       0]
 [      0       6    2832       0       0       0   94388]]
```

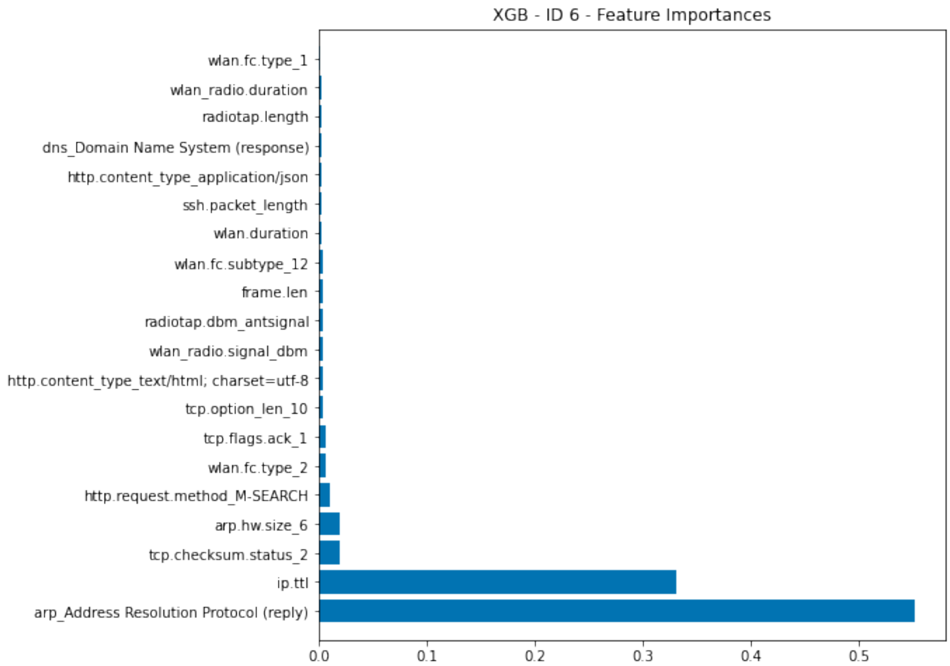


XGBoost Model 5 - Feature Importance

E.3.7 XGBoost Model 6 - Raw Metrics

```

Final Test Results
```

```
Test AUC: 99.99
Weighted Test F1: 99.65
Weighted Test Precision: 99.65
Weighted Test Recall: 99.65
Test Accuracy: 99.65

Classification Report
                      precision    recall  f1-score   support

      Botnet          0.96       0.74      0.84      17060
     Malware          0.86       0.85      0.86      39476
      Normal          1.00       1.00      1.00    3657765
         SQL          0.93       0.88      0.90        789
        SSDP          1.00       1.00      1.00    1649955
         SSH          0.95       0.79      0.86       3565
    WebSpoof          0.99       0.97      0.98     121533

    accuracy                                1.00    6404584
   macro avg          0.95       0.89      0.92    6404584
weighted avg          1.00       1.00      1.00    6404584

Confusion Matrix
[[   12703       31     4320        0        0        2        4]
 [      17    33596     5857        0        0        6        0]
 [     539     5289  4564546       56        0      144     1632]
 [       0        0       91      698        0        0        0]
 [       0        0        0        0  1649955        0        0]
 [       5       15      745        0        0     2800        0]
 [       1        4     3344        0        0        0   118184]]
```

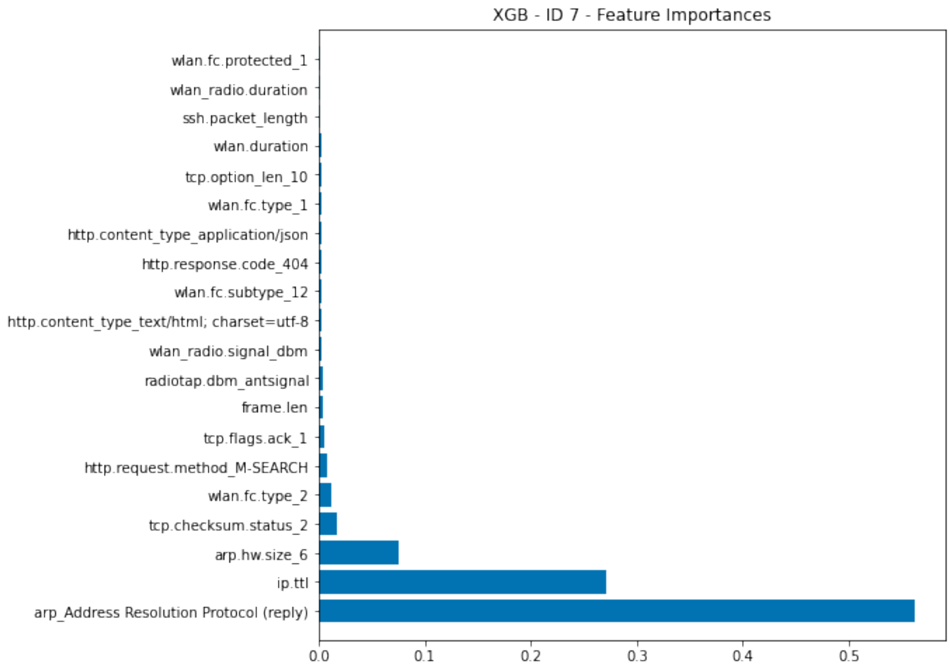


XGBoost Model 6 - Feature Importance

E.3.8 XGBoost Model 7 - Raw Metrics

```

Final Test Results
Test AUC: 99.99
Weighted Test F1: 99.65
Weighted Test Precision: 99.65
Weighted Test Recall: 99.65
Test Accuracy: 99.65

Classification Report
              precision    recall  f1-score   support

      Botnet       0.96      0.74      0.84     17060
     Malware       0.86      0.86      0.86     39476
      Normal       1.00      1.00      1.00   3657765
         SQL       0.93      0.89      0.91       789
        SSDP       1.00      1.00      1.00   1649955
         SSH       0.92      0.78      0.84      3565
    WebSpoof       0.99      0.97      0.98    121533

    accuracy                           1.00   6404584
   macro avg       0.95      0.89      0.92   6404584
weighted avg       1.00      1.00      1.00   6404584

Confusion Matrix
```

```
[[   12640      28    4387       0       0       2       3]
 [      17   34017    5434       0       0       8       0]
 [     546    5657 4564062      51       3     244    1643]
 [       0       0      87     702       0       0       0]
 [       0       0       0       0 1649955       0       0]
 [       3      14     760       0       0    2788       0]
 [       1      11    3341       0       0       0  118180]]
```



XGBoost Model 7 - Feature Importance

E.3.9 XGBoost Model 8 - Raw Metrics

```
S-CV Results


Final Test Results
Test AUC: 99.99
Weighted Test F1: 99.65
Weighted Test Precision: 99.65
Weighted Test Recall: 99.65
Test Accuracy: 99.65

Classification Report
              precision    recall  f1-score   support

```

```
       Botnet       0.96      0.74      0.84     17060
      Malware       0.86      0.86      0.86     39476
       Normal       1.00      1.00      1.00   4572206
          SQL       0.93      0.89      0.91       789
         SSDP       1.00      1.00      1.00   1649955
          SSH       0.92      0.78      0.85      3565
     WebSpoof       0.99      0.97      0.98    121533

     accuracy                           1.00   6404584
    macro avg       0.95      0.89      0.92   6404584
 weighted avg       1.00      1.00      1.00   6404584

Confusion Matrix
[[  12631      28     4396       0       0       2        3]
 [     17   34025     5426       0       0       8        0]
 [    539    5665  4564060      51       3     242     1646]
 [      0       0       89     700       0       0        0]
 [      0       0        0       0 1649955       0        0]
 [      3      14      755       0       0    2793        0]
 [      1      11     3308       0       0       0   118213]]
```

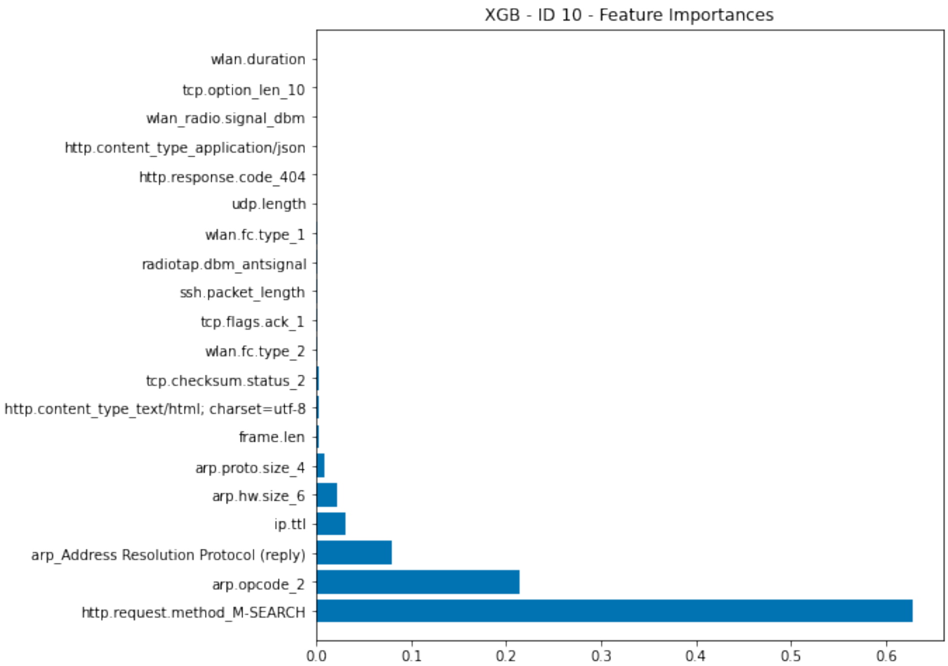
E.3.10 XGBoost Model 9 - Raw Metrics

```
RandomisedGridSearchResults
Time taken for CV: 150508.66 seconds
Best parameters found:
{'subsample': 0.9,
 'n_estimators': 200,
 'min_child_weight': 3,
 'max_depth': 9,
 'learning_rate': 0.3,
 'gamma': 0,
 'colsample_bytree': 0.7
}
```

E.3.11 XGBoost Model 10 - Raw Metrics

```
S-CV Results
Average AUC: 0.9999621491771731
Average F1-score: 0.9965027851331616
Average Precision: 0.9965087377065057
Average Recall: 0.9965808417525853
Average Accuracy: 0.9965808417525853

Final Test Results
Test ROC AUC:  99.98872586203336
```

```
Weighted Test F1:  99.65319267148135
Weighted Test Precision:  99.65352146874279
Weighted Test Recall:  99.66211700869252
Test Accuracy:  99.66211700869252

Classification Report
              precision    recall  f1-score   support

      Botnet       0.96      0.75      0.84     17060
     Malware       0.89      0.82      0.85     39476
      Normal       1.00      1.00      1.00   4572206
         SQL       0.94      0.89      0.91       789
        SSDP       1.00      1.00      1.00   1649955
         SSH       0.92      0.78      0.85      3565
    WebSpoof       0.99      0.97      0.98    121533

    accuracy                           1.00   6404584
   macro avg       0.96      0.89      0.92   6404584
weighted avg       1.00      1.00      1.00   6404584

Confusion Matrix
[[  12852      17    4183       0       0       3       5]
 [     18   32315    7132       0       0       8       3]
 [    580    3781 4565841      47       3     241    1713]
 [      0       0      89     700       0       0       0]
 [      0       0       0       0 1649955       0       0]
 [      6      15     749       0       0    2794       1]
 [      1       4    3041       0       0       0  118487]]
```

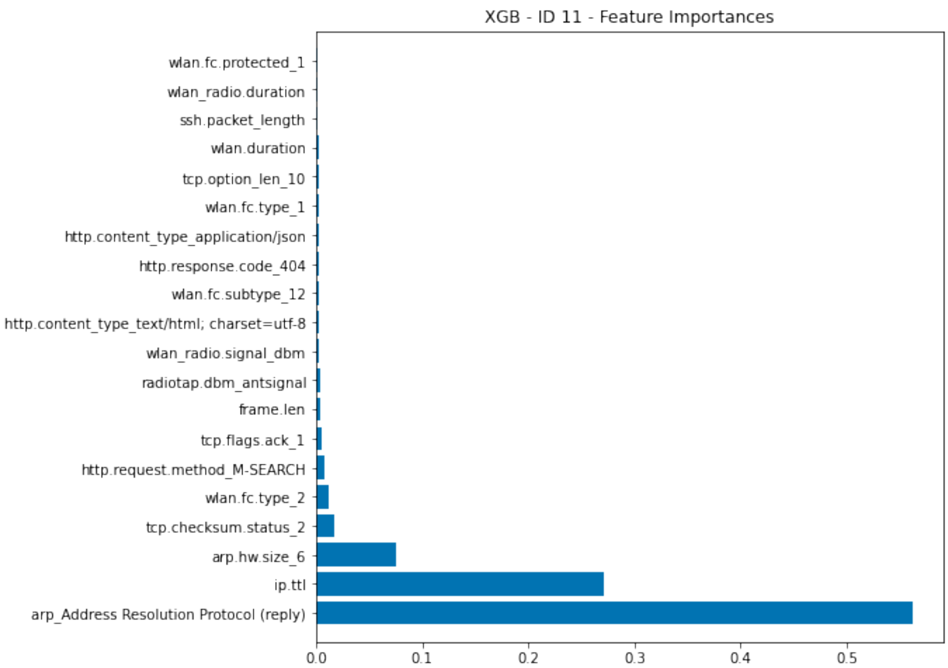


XGBoost Model 10 - Feature Importance

E.3.12 XGBoost Model 11 - Raw Metrics

```
S-CV Results
Average AUC: 99.99578159885314
Average F1-score: 99.64055055296684
Average Precision: 99.6431971824485
Average Recall: 99.6459656226469
Average Accuracy: 99.6459656226469
Training Time: 1450.04 seconds

Final Test Results
Test AUC:  99.98690081277596
Weighted Test F1:  99.64724140077732
Weighted Test Precision:  99.64992342160565
Weighted Test Recall:  99.65274871873021
Test Accuracy:  99.65274871873021

Classification Report
              precision    recall  f1-score   support

     Botnet        0.96      0.74      0.84     17060
    Malware        0.86      0.86      0.86     39476
     Normal        1.00      1.00      1.00   4572206
        SQL        0.93      0.89      0.91       789
       SSDP        1.00      1.00      1.00   1649955
        SSH        0.92      0.78      0.84      3565
```

```
      WebSpoof         0.99       0.97       0.98     121533

      accuracy                               1.00    6404584
     macro avg         0.95       0.89       0.92    6404584
  weighted avg         1.00       1.00       1.00    6404584

Confusion Matrix
[[   12640      28    4387       0       0       2       3]
 [      17   34017    5434       0       0       8       0]
 [     546    5657 4564062      51       3     244    1643]
 [       0       0      87     702       0       0       0]
 [       0       0       0       0 1649955       0       0]
 [       3      14     760       0       0    2788       0]
 [       1      11    3341       0       0       0  118180]]
```



XGBoost Model 11 - Feature Importance

F Neural Networks

F.0.1 MLP Model 0 - Raw Metrics

```
S-CV Results


Final Test Results
Test AUC: 99.9858021736145
Weighted Test F1: 99.3620514961493
Weighted Test Precision: 99.38432657598928
Weighted Test Recall: 99.40703938402462
Test Accuracy: 99.40703938402462

Classification Report

              precision    recall  f1-score   support

      Botnet       0.94      0.60      0.73     10236
     Malware       0.90      0.67      0.77     23686
      Normal       0.99      1.00      1.00   2743324
         SQL       1.00      0.08      0.15       473
        SSDP       1.00      1.00      1.00    989973
         SSH       0.89      0.46      0.61      2139
    WebSpoof       0.99      0.91      0.95     72920

    accuracy                           0.99   3842751
   macro avg       0.96      0.67      0.74   3842751
weighted avg       0.99      0.99      0.99   3842751

Confusion Matrix
[[   6106      48    4054       0       0      22       6]
 [     40   15809    7810       0       0      27       0]
 [    299    1707 2740823       0      10      67     418]
 [      0       0     434      39       0       0       0]
 [      0       0      10       0  989963       0       0]
 [      2      13    1140       0       0     984       0]
 [     74       0    6603       0       2       0   66241]]
```

F.0.2 MLP Model 1 - Raw Metrics

```
S-CV Results


Final Test Results

Test AUC: 0.9998605251312256
Weighted Test F1: 0.9934398041259884
Weighted Test Precision: 0.9935614162656269
```

```
Weighted Test Recall:  0.993817450050758
Test Accuracy:  0.993817450050758

Classification Report

              precision    recall  f1-score   support

      Botnet       0.87      0.58      0.70     10236
     Malware       0.88      0.71      0.79     23686
      Normal       0.99      1.00      1.00   2743324
         SQL       1.00      0.38      0.55       473
        SSDP       1.00      1.00      1.00    989973
         SSH       0.91      0.45      0.60      2139
    WebSpoof       0.99      0.89      0.94     72920

    accuracy                           0.99   3842751
   macro avg       0.95      0.72      0.80   3842751
weighted avg       0.99      0.99      0.99   3842751

Confusion Matrix

[[   5977      36    4205       0       0      14       4]
 [    219   16928    6531       0       0       8       0]
 [    603    2233 2740011       0       9      74     394]
 [      2       5     282     181       0       3       0]
 [      0       0       9       0  989964       0       0]
 [      1      22    1151       0       0     965       0]
 [     30      12    7909       0       2       0   64967]]
```

F.0.3 MLP Model 2 - Raw Metrics

```

Final Test Results

Test AUC:  0.9986337212130303
Weighted Test Precision:  0.9942064571755157
Weighted Test Recall:  0.9943597037043976
Weighted Test F1:  0.9939107235001085
Test Accuracy:  0.9943597037043976

Classification Report

              precision    recall  f1-score   support

      Botnet       0.93      0.62      0.74     13648
     Malware       0.96      0.64      0.77     31581
      Normal       0.99      1.00      1.00   3657765
         SQL       1.00      0.13      0.23       631
        SSDP       1.00      1.00      1.00   1319964
         SSH       0.84      0.57      0.68      2852
```

```
      WebSpoof          1.00        0.91        0.95      97226

     accuracy                                  0.99    5123667
    macro avg           0.96        0.70        0.77    5123667
 weighted avg           0.99        0.99        0.99    5123667

Confusion Matrix

[[    8403      163     5018        0        0       26       38]
 [      36    20332    11185        0        0       28        0]
 [     550      770  3655857        0       12      258      318]
 [       0        0      548       83        0        0        0]
 [       0        0       12        0  1319952        0        0]
 [      12        0     1222        0        0     1618        0]
 [      33       16     8653        0        1        0    88523]]
```

F.0.4 MLP Model 3 - Raw Metrics

```

Final Test Results

Test AUC: 0.9998477896054586
Weighted Test F1:  0.9939352564175636
Weighted Test Precision:  0.9943562023089472
Weighted Test Recall:  0.9944416762447676
Test Accuracy:  0.9944416762447676

Classification Report

                precision      recall   f1-score    support

       Botnet           0.92        0.63        0.75      13648
      Malware           0.99        0.62        0.76      31581
       Normal           0.99        1.00        1.00    3657765
          SQL           1.00        0.15        0.25        631
         SSDP           1.00        1.00        1.00    1319964
          SSH           0.93        0.45        0.61       2852
     WebSpoof           1.00        0.92        0.95      97226

     accuracy                                  0.99    5123667
    macro avg           0.98        0.68        0.76    5123667
 weighted avg           0.99        0.99        0.99    5123667

Confusion Matrix

[[    8644       40     4952        0        0        2       10]
  [     295    19427    11852        0        0        7        0]
  [     409      246  3656779        0        9       84      238]
  [       0        0      538       92        1        0        0]
  [       0        0       16        0  1319948        0        0]
```

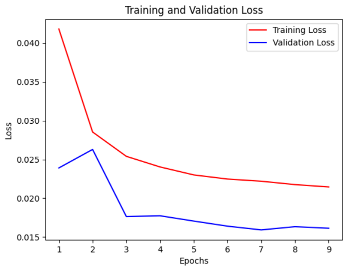
```
[        5         0      1569         0         0      1278         0]
[       25         2      8178         0         1         0     89020]]
```

F.0.5 MLP Model 4 - Raw Metrics

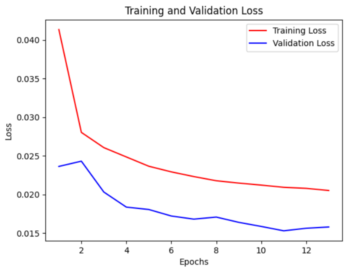
```
S-CV Results
Average AUC: 99.90
Average F1: 99.37
Average Precision: 99.40
Average Recall: 99.42
Average Accuracy: 99.42
CV Time 2007.377500295639 seconds

Final Test Results
Test AUC:  0.9993877331212752
Weighted Test F1:  0.9942475033645454
Weighted Test Precision:  0.9943763031781035
Weighted Test Recall:  0.994580131980469
Test Accuracy:  0.994580131980469

Classification Report
              precision    recall  f1-score   support

      Botnet       0.94      0.61      0.74     17060
     Malware       0.89      0.72      0.80     39476
      Normal       0.99      1.00      1.00   4572206
         SQL       0.99      0.37      0.54       789
        SSDP       1.00      1.00      1.00   1649955
         SSH       0.83      0.48      0.60      3565
   WebSpoof       1.00      0.92      0.95    121533

    accuracy                           0.99   6404584
   macro avg       0.95      0.73      0.80   6404584
weighted avg       0.99      0.99      0.99   6404584

Confusion Matrix
[[   10357      139     6453        0        0      101       10]
 [     112    28556    10769        0        0       39        0]
 [     507     3351  4567732        2        8      205      401]
 [       0        0      495      294        0        0        0]
 [       0        0       10        0  1649945        0        0]
 [       0       28     1842        0        0     1695        0]
 [      34       11    10194        0        1        0   111293]]
```

Figure F.1: MLP Model 4 - Loss Curves

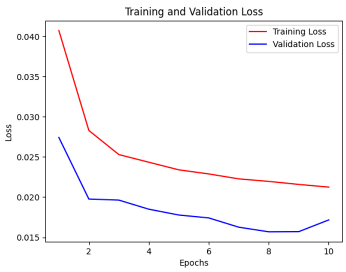
(a) Fold 1



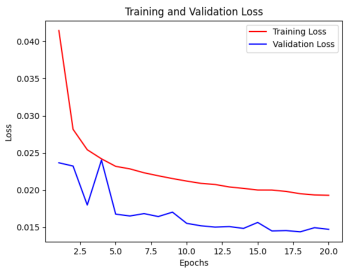
(b) Fold 2



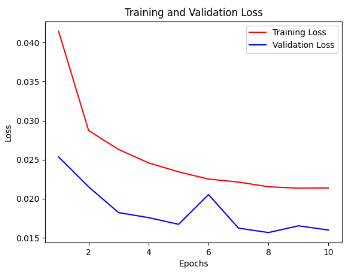
(c) Fold 3



(d) Fold 4

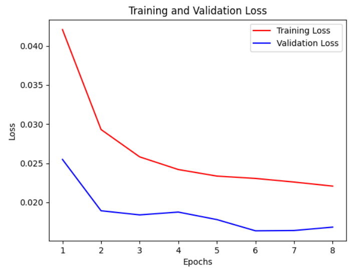


(e) Fold 5

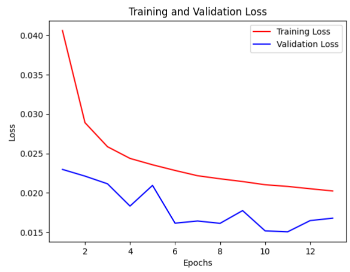


(f) Fold 6

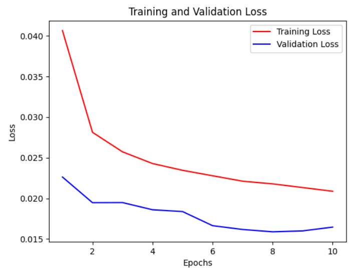
(g) Fold 7



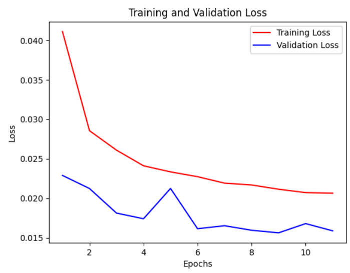
(h) Fold 8



(i) Fold 9



(j) Fold 10



F.0.6 MLP Model 5 - Raw Metrics

```
S-CV Results
AUC: 0.9840
F1: 0.9468
Precision: 0.9685
Recall: 0.9549
Accuracy: 0.9549
CV Time: 1892.0350830554962 seconds

Final Test Results
Test AUC: 0.9988450838388412
Weighted Test F1: 0.9935567670757577
Weighted Test Precision: 0.993714141198025
Weighted Test Recall: 0.9940441096564585
Test Accuracy: 0.9940441096564585

Classification Report

                 precision    recall  f1-score   support

      Botnet        0.87      0.54      0.66     17060
     Malware        0.90      0.64      0.75     39476
      Normal        0.99      1.00      1.00   4572206
         SQL        0.99      0.34      0.51       789
        SSDP        1.00      1.00      1.00   1649955
         SSH        0.80      0.47      0.59      3565
    WebSpoof        0.99      0.92      0.95    121533

    accuracy                            0.99   6404584
   macro avg        0.94      0.70      0.78   6404584
weighted avg        0.99      0.99      0.99   6404584

Confusion Matrix
[[   9180    1338    6411       0       0     124       7]
 [      9   25233   14198       0       0      36       0]
 [   1206    1337 4568726       3      20     258     656]
 [      0       0     513     270       2       4       0]
 [      0       0      10       0 1649945       0       0]
 [      0       1    1877       0       0    1687       0]
 [    196       1    9935       0       3       0  111398]]
```

F.0.7 MLP Model 6 - Raw Metrics

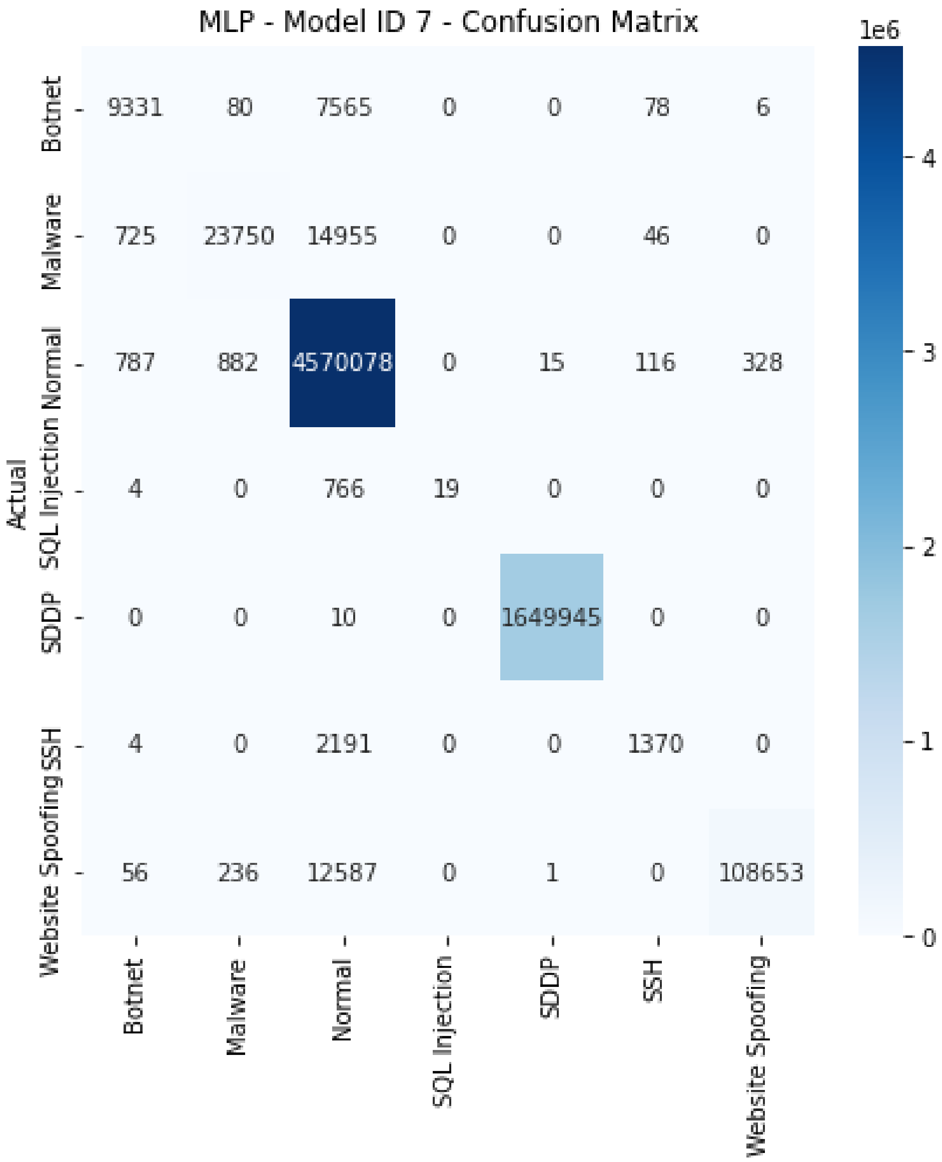
```
S-CV Results
AUC: 0.9979
F1: 0.9927
Precision: 0.9931
Recall: 0.9933
```

```
Accuracy: 0.9933
CV Time: 4323.574688911438 seconds

Final Test Results
Test AUC: 0.9979510725501038
Weighted Test F1: 0.9937794324941432
Weighted Test Precision: 0.9939653920793858
Weighted Test Recall: 0.9941811989662405
Test Accuracy: 0.9941811989662405

Classification Report
              precision    recall  f1-score   support

      Botnet       0.96      0.52      0.68     17060
     Malware       0.83      0.72      0.77     39476
      Normal       0.99      1.00      1.00   4572206
         SQL       0.98      0.13      0.22       789
        SSDP       1.00      1.00      1.00   1649955
         SSH       0.91      0.47      0.62      3565
    WebSpoof       0.98      0.94      0.96    121533

    accuracy                           0.99   6404584
   macro avg       0.95      0.68      0.75   6404584
weighted avg       0.99      0.99      0.99   6404584

Confusion Matrix
[[    8899     195    7823       0       0      17     126]
 [      43   28231   11177       0       0      25       0]
 [     244    4724 4564566       2      12     122    2536]
 [       0       0     689     100       0       0       0]
 [       0       0      10       0 1649945       0       0]
 [       0     519    1388       0       0    1658       0]
 [      62     237    7315       0       1       0  113918]]
```

F.0.8 MLP Model 7 - Raw Metrics

```
S-CV Results
AUC: 99.72
F1: 99.23
Precision: 99.25
Recall: 99.31
Accuracy: 99.31
CV Time: 3009.0557339191437 seconds

Final Test Results
Test AUC: 99.8447719823538
Weighted Test F1: 99.28992200626915
Weighted Test Precision: 99.32770674412539
Weighted Test Recall: 99.35299466756935
Test Accuracy: 99.35299466756935
```

```

Classification Report
              precision    recall  f1-score   support

           0       0.86      0.55      0.67     17060
           1       0.95      0.60      0.74     39476
           2       0.99      1.00      1.00   4572206
           3       1.00      0.02      0.05       789
           4       1.00      1.00      1.00   1649955
           5       0.85      0.38      0.53      3565
           6       1.00      0.89      0.94    121533

    accuracy                           0.99   6404584
   macro avg       0.95      0.64      0.70   6404584
weighted avg       0.99      0.99      0.99   6404584

Confusion Matrix
[[   9331      80    7565       0       0      78       6]
 [    725   23750   14955       0       0      46       0]
 [    787     882 4570078       0      15     116     328]
 [      4       0     766      19       0       0       0]
 [      0       0      10       0 1649945       0       0]
 [      4       0    2191       0       0    1370       0]
 [     56     236   12587       0       1       0  108653]]
```



MLP Model 7 - CM

G Model Code

The following contains sections of the code relevant to creating the models. Only the 'best models' were included in this section to reduce repeating code and reduce the size of the appendices. The complete code for every model can be found in the code.zip and the full code base.

G.1 Data Cleaning

The following section contains the steps required to process the individual datasets for each class.

Botnet

```
import pandas as pd
import re

# Define the columns to keep
cols_to_use = ['frame.len', 'radiotap.dbm_antsignal', 'radiotap.length', 'wlan.duration',
               'wlan_radio.duration', 'wlan_radio.signal_dbm', 'radiotap.present.tsft',
               'wlan.fc.type', 'wlan.fc.subtype', 'wlan.fc.ds', 'wlan.fc.frag',
               'wlan.fc.moredata', 'wlan.fc.protected', 'wlan.fc.pwrmgt', 'wlan.fc.retry',
               'wlan_radio.phy', 'udp.length', 'ip.ttl', 'arp', 'arp.proto.type',
               'arp.hw.size', 'arp.proto.size', 'arp.hw.type', 'arp.opcode',
               'tcp.analysis', 'tcp.analysis.retransmission', 'tcp.option_len',
               'tcp.checksum.status', 'tcp.flags.ack', 'tcp.flags.fin', 'tcp.flags.push',
               'tcp.flags.reset', 'tcp.flags.syn', 'dns', 'dns.count.queries', 'dns.count.answers',
               'dns.resp.len', 'dns.resp.ttl', 'http.request.method', 'http.response.code',
               'http.content_type', 'ssh.message_code', 'ssh.packet_length', 'nbns',
               'nbss.length', 'nbss.type', 'ldap', 'smb2.cmd', 'smb.flags.response',
               'smb.access.generic_read', 'smb.access.generic_write', 'smb.access.generic_execute','Label']

# Define the chunk size to read in each pass
batch_size = 1000000

combined_df = pd.DataFrame()

# Iterate through the file in batches
for chunk in pd.read_csv('botnet_reduced.csv', chunksize=batch_size, usecols=cols_to_use, low_memory=False):

    # Combine the processed chunk with previous chunks
    combined_df = pd.concat([combined_df, chunk])

# Drop all missing rows that contain only nan values
combined_df = combined_df.dropna(how='all')
```

```

# Drop all rows with missing values in Label Column
combined_df = combined_df.dropna(subset=['Label'])


# Fill NAs with zeros
# Change nan values to 0
combined_df = combined_df.fillna(0)

# Duplicate the df
df = combined_df.copy()

# Regex to keep only the first value e.g
# -100-100-10 becomes -100,  123-456-1 becomes 123, -10-2
    becomes-10, 81-63-63 becomes 81
def seperated_values(x):
    x = str(x)
    match = re.match(r'^(-?\d+).*$', x)
    if match:
        return match.group(1)
    else:
        return x

# Go through all columns and change separate values into just
    one value
for column in df.columns:
    df[column] = df[column].apply(seperated_values)
    print('Processing', column)
print('Done')


# 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.
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"

# Use str.match method to apply the regex pattern to the column
mask = df['tcp.option_len'].astype(str).str.match(regex).fillna(
    False)
# Filter the dataframe
df = df[~mask]

# Use str.match method to apply the regex pattern to the column
mask = df['dns.resp.ttl'].astype(str).str.match(regex).fillna(
    False)
df = df[~mask]

# Use str.match method to apply the regex pattern to the column
mask = df['ip.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask]

# Use str.match method to apply the regex pattern to the column
```

```
mask = df['smb2.cmd'].astype(str).str.match(regex).fillna(False)
df = df[~mask]

df.to_csv('botnet.csv', index=False)
```

Malware

```
import pandas as pd
import re

# Define the columns to keep
cols_to_use = ['frame.len', 'radiotap.dbm_antsignal', 'radiotap.length', 'wlan.duration',
               'wlan_radio.duration', 'wlan_radio.signal_dbm', 'radiotap.present.tsft',
               'wlan.fc.type', 'wlan.fc.subtype', 'wlan.fc.ds', 'wlan.fc.frag',
               'wlan.fc.moredata', 'wlan.fc.protected', 'wlan.fc.pwrmgt', 'wlan.fc.retry',
               'wlan_radio.phy', 'udp.length', 'ip.ttl', 'arp', 'arp.proto.type',
               'arp.hw.size', 'arp.proto.size', 'arp.hw.type', 'arp.opcode',
               'tcp.analysis', 'tcp.analysis.retransmission', 'tcp.option_len',
               'tcp.checksum.status', 'tcp.flags.ack', 'tcp.flags.fin', 'tcp.flags.push',
               'tcp.flags.reset', 'tcp.flags.syn', 'dns', 'dns.count.queries', 'dns.count.answers',
               'dns.resp.len', 'dns.resp.ttl', 'http.request.method', 'http.response.code',
               'http.content_type', 'ssh.message_code', 'ssh.packet_length', 'nbns',
               'nbss.length', 'nbss.type', 'ldap', 'smb2.cmd', 'smb.flags.response',
               'smb.access.generic_read', 'smb.access.generic_write', 'smb.access.generic_execute','Label']

# Define the chunk size to read in each pass
batch_size = 1000000

combined_df = pd.DataFrame()

# Iterate through the file in batches
for chunk in pd.read_csv('malware_reduced.csv', chunksize=batch_size, usecols=cols_to_use, low_memory=False):

    # Combine the processed chunk with previous chunks
    combined_df = pd.concat([combined_df, chunk])

```

```python
# Drop all missing rows that contain only nan values
combined_df = combined_df.dropna(how='all')

# Drop all rows with missing values in Label Column
combined_df = combined_df.dropna(subset=['Label'])

# Fill NAs with zeros
# Change nan values to 0
combined_df = combined_df.fillna(0)

# Duplicate the dataframe
df = combined_df.copy()

# ## Seperate Hyphen Values

# Regex to keep only the first value e.g
# -100-100-10 becomes -100,   123-456-1 becomes 123, -10-2 becomes-10, 81-63-63 becomes 81
def seperated_values(x):
    x = str(x)
    match = re.match(r'^(-?\d+).*$', x)
    if match:
        return match.group(1)
    else:
        return x

# Go through all columns and change seperate values into just one value
for column in df.columns:
    df[column] = df[column].apply(seperated_values)
    print('Processing', column)
print('Done')

df['Label'].value_counts()
df['tcp.option_len'].value_counts()
df['ip.ttl'].value_counts()
df['dns.resp.ttl'].value_counts()

# 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.
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"

# ### tcp.option_len
# Use str.match method to apply the regex pattern to the column
mask = df['tcp.option_len'].astype(str).str.match(regex).fillna(False)
df = df[~mask] # Filter the dataframe
df['tcp.option_len'].value_counts()

# ### dns.resp.ttl
# Use str.match method to apply the regex pattern to the column
```

```
mask = df['dns.resp.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['dns.resp.ttl'].value_counts()

# ### ip.ttl
# Use str.match method to apply the regex pattern to the column
mask = df['ip.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask] # Filter the dataframe
df['ip.ttl'].value_counts()

# ### smb2.cmd
# Use str.match method to apply the regex pattern to the column
mask = df['smb2.cmd'].astype(str).str.match(regex).fillna(False)
df = df[~mask] # Filter the dataframe
df['smb2.cmd'].value_counts()

# ## Export to CSV
df.to_csv('malware.csv', index=False)
```

SQL

```
import pandas as pd
import re

# Define the columns to keep
cols_to_use = ['frame.len', 'radiotap.dbm_antsignal', 'radiotap.length', 'wlan.duration',
               'wlan_radio.duration', 'wlan_radio.signal_dbm', 'radiotap.present.tsft',
               'wlan.fc.type', 'wlan.fc.subtype', 'wlan.fc.ds', 'wlan.fc.frag',
               'wlan.fc.moredata', 'wlan.fc.protected', 'wlan.fc.pwrmgt', 'wlan.fc.retry',
               'wlan_radio.phy', 'udp.length', 'ip.ttl', 'arp', 'arp.proto.type',
               'arp.hw.size', 'arp.proto.size', 'arp.hw.type', 'arp.opcode',
               'tcp.analysis', 'tcp.analysis.retransmission', 'tcp.option_len',
               'tcp.checksum.status', 'tcp.flags.ack', 'tcp.flags.fin', 'tcp.flags.push',
               'tcp.flags.reset', 'tcp.flags.syn', 'dns', 'dns.count.queries', 'dns.count.answers',
               'dns.resp.len', 'dns.resp.ttl', 'http.request.method', 'http.response.code',
               'http.content_type', 'ssh.message_code', 'ssh.packet_length', 'nbns',
               'nbss.length', 'nbss.type', 'ldap', 'smb2.cmd', 'smb.flags.response',
```

```
                     'smb.access.generic_read', 'smb.access.generic_write', 'smb.access.generic_execute','Label']

# Define the chunk size to read in each pass
batch_size = 1000000

combined_df = pd.DataFrame()

# Iterate through the file in batches
for chunk in pd.read_csv('sql_reduce.csv', chunksize=batch_size, usecols=cols_to_use, low_memory=False):

    # Combine the processed chunk with previous chunks
    combined_df = pd.concat([combined_df, chunk])

combined_df.info()

# Check for missing values
combined_df.isna().sum()

# Drop all missing rows that contain only nan values
combined_df = combined_df.dropna(how='all')

# Drop all rows with missing values in Label Column
combined_df = combined_df.dropna(subset=['Label'])

# Fill NAs with zeros
# Change nan values to 0
combined_df = combined_df.fillna(0)

# Duplicate the dataframe
df = combined_df.copy()
df.info()

# ## Seperate Hyphen Values

# Regex to keep only the first value e.g
# -100-100-10 becomes -100,   123-456-1 becomes 123, -10-2 becomes-10, 81-63-63 becomes 81
def seperated_values(x):
    x = str(x)
    match = re.match(r'^(-?\d+).*$', x)
    if match:
        return match.group(1)
    else:
        return x

# Go through all columns and change separate values into just one value
for column in df.columns:
    df[column] = df[column].apply(seperated_values)
    print('Processing', column)
print('Done')

```

```
df['Label'].value_counts()
df['tcp.option_len'].value_counts()
df['ip.ttl'].value_counts()
df['dns.resp.ttl'].value_counts()
# 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.
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"

# ### tcp.option_len
# Use str.match method to apply the regex pattern to the column
mask = df['tcp.option_len'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['tcp.option_len'].value_counts()

# ### dns.resp.ttl
# Use str.match method to apply the regex pattern to the column
mask = df['dns.resp.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['dns.resp.ttl'].value_counts()

# ### ip.ttl
# Use str.match method to apply the regex pattern to the column
mask = df['ip.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['ip.ttl'].value_counts()

# ### smb2.cmd
# Use str.match method to apply the regex pattern to the column
mask = df['smb2.cmd'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['smb2.cmd'].value_counts()

# ## Export to CSV
df.to_csv('sql_reduced.csv', index=False)
```

SSDP

```
import pandas as pd
import re
from sklearn.model_selection import train_test_split

# Define the columns to keep
cols_to_use = ['frame.len', 'radiotap.dbm_antsignal', 'radiotap.length', 'wlan.duration',
```

```
                       'wlan_radio.duration', 'wlan_radio.signal_dbm', 'radiotap.present.tsft',
                       'wlan.fc.type', 'wlan.fc.subtype', 'wlan.fc.ds', 'wlan.fc.frag',
                       'wlan.fc.moredata', 'wlan.fc.protected', 'wlan.fc.pwrmgt', 'wlan.fc.retry',
                       'wlan_radio.phy', 'udp.length', 'ip.ttl', 'arp', 'arp.proto.type',
                       'arp.hw.size', 'arp.proto.size', 'arp.hw.type', 'arp.opcode',
                       'tcp.analysis', 'tcp.analysis.retransmission', 'tcp.option_len',
                       'tcp.checksum.status', 'tcp.flags.ack', 'tcp.flags.fin', 'tcp.flags.push',
                       'tcp.flags.reset', 'tcp.flags.syn', 'dns', 'dns.count.queries', 'dns.count.answers',
                       'dns.resp.len', 'dns.resp.ttl', 'http.request.method', 'http.response.code',
                       'http.content_type', 'ssh.message_code', 'ssh.packet_length', 'nbns',
                       'nbss.length', 'nbss.type', 'ldap', 'smb2.cmd', 'smb.flags.response',
                       'smb.access.generic_read', 'smb.access.generic_write', 'smb.access.generic_execute','Label']

# Define the chunk size to read in each pass
batch_size = 1000000

combined_df = pd.DataFrame()

# Iterate through the file in batches
for chunk in pd.read_csv('ssdp_combined.csv', chunksize=batch_size, usecols=cols_to_use, low_memory=False):

    # Combine the processed chunk with previous chunks
    combined_df = pd.concat([combined_df, chunk])

combined_df.info()
combined_df.isna().sum()

# Drop all missing rows that contain only nan values
combined_df = combined_df.dropna(how='all')

# Drop all rows with missing values in Label Column
combined_df = combined_df.dropna(subset=['Label'])

combined_df = combined_df.fillna(0) # Change nan values to 0
df = combined_df.copy() # Duplicate the dataframe

df = df.reindex(columns=cols_to_use)

# ## Decimal To Int

```

```
# Convert any floats that contain no decimals into integer datatypes

# Define the column name to search for decimals
column_name = 'wlan.duration'
# Create a regular expression pattern to match numbers with decimals
decimal_pattern = r'\d+\.\d+'

# Count the number of matches in the specified column
decimal_count = 0
for value in df[column_name]:
    if isinstance(value, str) and re.search(decimal_pattern, value):
        decimal_count += 1

# Print the result
print(f"The number of rows with decimals in column '{column_name}' is {decimal_count}.")

df[['radiotap.length','ip.ttl']] = df[['radiotap.length', 'ip.ttl'[].astype(int)
#df['ip.ttl'] = df['ip.ttl'].astype(int)
#df['wlan.duration'] = df['wlan.duration'].astype(int)

# ## Seperate Hyphen Values

# Regex to keep only the first value e.g
# -100-100-10 becomes -100,  123-456-1 becomes 123, -10-2 becomes-10, 81-63-63 becomes 81
def seperated_values(x):
    x = str(x)
    match = re.match(r'^(-?\d+).*$', x)
    if match:
        return match.group(1)
    else:
        return x

# Go through all columns and change seperate values into just one value
for column in df.columns:
    df[column] = df[column].apply(seperated_values)
    print('Processing', column)

df['tcp.option_len'].value_counts()

# 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.
#regex = r"\b(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)-\d{2}\b"
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"
```

```
for index, row in df.iterrows():
    # If the value is a string and matches the regex
    if isinstance(row['tcp.option_len'], str) and re.search(
    regex, row['tcp.option_len']):
        #print(row['tcp.option_len'])
        df.drop(index, inplace=True) # Drop the value in place
    by its index

for index, row in df.iterrows():
    # If the value is a string and matches the regex
    if isinstance(row['ip.ttl'], str) and re.search(regex, row['
    ip.ttl']):
        df.drop(index, inplace=True) # Drop the value in place
    by its index

for index, row in df.iterrows():
    # If the value is a string and matches the regex
    if isinstance(row['dns.resp.ttl'], str) and re.search(regex,
     row['dns.resp.ttl']):
        df.drop(index, inplace=True) # Drop the value in place
    by its index

# Use str.match method to apply the regex pattern to the column
mask = df['tcp.option_len'].astype(str).str.match(regex).fillna(
    False)

# Filter the dataframe
df = df[~mask]

df['tcp.option_len'].value_counts()

# Use str.match method to apply the regex pattern to the column
mask = df['dns.resp.ttl'].astype(str).str.match(regex).fillna(
    False)

# Filter the dataframe
df = df[~mask]

df['dns.resp.ttl'].value_counts()

# Use str.match method to apply the regex pattern to the column
mask = df['ip.ttl'].astype(str).str.match(regex).fillna(False)

# Filter the dataframe
df = df[~mask]

df['ip.ttl'].value_counts()
df['Label'] = df['Label'].replace({'SDDP': 'SSDP'})

df.to_csv('sddp.csv', index=False)

# # Testing
```

```
combined_df.to_csv('ssdp_reduced.csv', index=False)
batch_size = 1000000
combined_df = pd.DataFrame()

# Iterate through the file in batches
for chunk in pd.read_csv('ssdp_reduced.csv', chunksize=
    batch_size, low_memory=False):

    # Combine the processed chunk with previous chunks
    combined_df = pd.concat([combined_df, chunk])

combined_df.head()

combined_df['radiotap.dbm_antsignal'].value_counts()

processed_df = combined_df.copy()
processed_df['radiotap.dbm_antsignal'] = combined_df['radiotap.
    dbm_antsignal'].apply(seperated_values)
processed_df['dns.resp.ttl'] = combined_df['dns.resp.ttl'].apply
    (seperated_values)
processed_df['udp.length'] = combined_df['udp.length'].apply(
    seperated_values)
processed_df['ip.ttl'] = combined_df['ip.ttl'].apply(
    seperated_values)
processed_df['dns.count.answers'] = combined_df['dns.count.
    answers'].apply(seperated_values)
processed_df['nbss.length'] = combined_df['nbss.length'].apply(
    seperated_values)
processed_df['smb2.cmd'] = combined_df['smb2.cmd'].apply(
    seperated_values)
processed_df['ssh.packet_length'] = combined_df['ssh.
    packet_length'].apply(seperated_values)
processed_df['ssh.message_code'] = combined_df['ssh.message_code
    '].apply(seperated_values)
combined_df['ip.ttl'].value_counts()
processed_df['ip.ttl'].value_counts()

# Define the column name to search for decimals
column_name = 'ip.ttl'

# Create a regular expression pattern to match numbers with
    decimals
decimal_pattern = r'\d+\.\d+'

# Count the number of matches in the specified column
decimal_count = 0
for value in combined_df[column_name]:
    if isinstance(value, str) and re.search(decimal_pattern,
    value):
        print(value)
        decimal_count += 1

# Print the result
```

```
print(f"The number of rows with decimals in column '{column_name}' is {decimal_count}.")

def separated_values(x):
    # Convert x to a string and remove any leading/trailing whitespaces
    x = str(x).strip()
    # Use regex to check if x contains any non-digit character
    if re.search(r'\D', x):
        # x contains non-digit character, so return None to mark for removal
        return None
    else:
        # x contains only digits, so apply the original separated_values logic
        match = re.match(r'^(-?\d+).*$', x)
        if match:
            print(match.group(1))
            return match.group(1)
        else:
            return x

processed_df['ip.ttl'] = combined_df['ip.ttl'].apply(seperated_values)
#processed_df = processed_df.dropna()

# Change Data Types

processed_df['udp.length'] = processed_df['udp.length'].astype(float).astype(int)
processed_df.head()

# ### Subset of data
processed_df['Label'].value_counts()
processed_df.isna().sum()

#  Encode the target variable, 0 is Normal, 1 is SSH, 2 is SQL Injection
processed_df['Label'] = processed_df['Label'].map({'Normal': 0, 'SDDP': 1})

# Split the data into training and testing sets, preserving the class distribution
train_data, test_data, train_labels, test_labels = train_test_split(processed_df.drop('Label', axis=1), processed_df['Label'], test_size=0.5, stratify=processed_df['Label'])

train_data.value_counts()
# Combine the training data and labels
reduced_df = pd.concat([train_data, train_labels], axis=1)

# Export the subset of data to a CSV file
reduced_df.to_csv('ssdp_reduced.csv', index=False)
```

SSH

```python
import pandas as pd
import re

# Define the columns to keep
cols_to_use = ['frame.len', 'radiotap.dbm_antsignal', 'radiotap.length', 'wlan.duration',
               'wlan_radio.duration', 'wlan_radio.signal_dbm', 'radiotap.present.tsft',
               'wlan.fc.type', 'wlan.fc.subtype', 'wlan.fc.ds', 'wlan.fc.frag',
               'wlan.fc.moredata', 'wlan.fc.protected', 'wlan.fc.pwrmgt', 'wlan.fc.retry',
               'wlan_radio.phy', 'udp.length', 'ip.ttl', 'arp', 'arp.proto.type',
               'arp.hw.size', 'arp.proto.size', 'arp.hw.type', 'arp.opcode',
               'tcp.analysis', 'tcp.analysis.retransmission', 'tcp.option_len',
               'tcp.checksum.status', 'tcp.flags.ack', 'tcp.flags.fin', 'tcp.flags.push',
               'tcp.flags.reset', 'tcp.flags.syn', 'dns', 'dns.count.queries', 'dns.count.answers',
               'dns.resp.len', 'dns.resp.ttl', 'http.request.method', 'http.response.code',
               'http.content_type', 'ssh.message_code', 'ssh.packet_length', 'nbns',
               'nbss.length', 'nbss.type', 'ldap', 'smb2.cmd', 'smb.flags.response',
               'smb.access.generic_read', 'smb.access.generic_write', 'smb.access.generic_execute','Label']

# Define the chunk size to read in each pass
batch_size = 1000000

combined_df = pd.DataFrame()

# Iterate through the file in batches
for chunk in pd.read_csv('ssh_reduced.csv', chunksize=batch_size, usecols=cols_to_use, low_memory=False):

    # Combine the processed chunk with previous chunks
    combined_df = pd.concat([combined_df, chunk])

combined_df.info()

# Check for missing values
```

```python
combined_df.isna().sum()

# Drop all missing rows that contain only nan values
combined_df = combined_df.dropna(how='all')

# Drop all rows with missing values in Label Column
combined_df = combined_df.dropna(subset=['Label'])

# Fill NAs with zeros
# Change nan values to 0
combined_df = combined_df.fillna(0)

# Duplicate the dataframe
df = combined_df.copy()

df.info()

# Regex to keep only the first value e.g
# -100-100-10 becomes -100,  123-456-1 becomes 123, -10-2 becomes-10, 81-63-63 becomes 81
def seperated_values(x):
    x = str(x)
    match = re.match(r'^(-?\d+).*$', x)
    if match:
        return match.group(1)
    else:
        return x

# Go through all columns and change separate values into just one value
for column in df.columns:
    df[column] = df[column].apply(seperated_values)
    print('Processing', column)
print('Done')

df['Label'].value_counts()
df['tcp.option_len'].value_counts()
df['ip.ttl'].value_counts()
df['dns.resp.ttl'].value_counts()

# 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.
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"

# ### tcp.option_len
# Use str.match method to apply the regex pattern to the column
mask = df['tcp.option_len'].astype(str).str.match(regex).fillna(False)
# Filter the dataframe
df = df[~mask]
df['tcp.option_len'].value_counts()
```

```

# ### dns.resp.ttl
# Use str.match method to apply the regex pattern to the column
mask = df['dns.resp.ttl'].astype(str).str.match(regex).fillna(
    False)
df = df[~mask]
df['dns.resp.ttl'].value_counts()


# ### ip.ttl
mask = df['ip.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['ip.ttl'].value_counts()


# ### smb2.cmd
mask = df['smb2.cmd'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['smb2.cmd'].value_counts()

# ## Export to CSV
df.to_csv('ssh.csv', index=False)
```

Website Spoofing

```
import pandas as pd
import re

# Define the columns to keep
cols_to_use = ['frame.len', 'radiotap.dbm_antsignal', 'radiotap.
    length', 'wlan.duration',
                'wlan_radio.duration', 'wlan_radio.signal_dbm
    ', 'radiotap.present.tsft',
                'wlan.fc.type', 'wlan.fc.subtype', 'wlan.fc.
    ds', 'wlan.fc.frag',
                'wlan.fc.moredata', 'wlan.fc.protected', '
    wlan.fc.pwrmgt', 'wlan.fc.retry',
                'wlan_radio.phy', 'udp.length', 'ip.ttl', '
    arp', 'arp.proto.type',
                'arp.hw.size', 'arp.proto.size', 'arp.hw.type
    ', 'arp.opcode',
                'tcp.analysis', 'tcp.analysis.retransmission'
    , 'tcp.option_len',
                'tcp.checksum.status', 'tcp.flags.ack', 'tcp.
    flags.fin', 'tcp.flags.push',
                'tcp.flags.reset', 'tcp.flags.syn', 'dns', '
    dns.count.queries', 'dns.count.answers',
                'dns.resp.len', 'dns.resp.ttl', 'http.request
    .method', 'http.response.code',
```

```
                    'http.content_type', 'ssh.message_code', 'ssh.packet_length', 'nbns',
                    'nbss.length', 'nbss.type', 'ldap', 'smb2.cmd', 'smb.flags.response',
                    'smb.access.generic_read', 'smb.access.generic_write', 'smb.access.generic_execute','Label']

# Define the chunk size to read in each pass
batch_size = 1000000

combined_df = pd.DataFrame()

# Iterate through the file in batches
for chunk in pd.read_csv('website_spoofing_reduced.csv', chunksize=batch_size, usecols=cols_to_use, low_memory=False):
    # Combine the processed chunk with previous chunks
    combined_df = pd.concat([combined_df, chunk])

combined_df.info()
# Check for missing values
combined_df.isna().sum()

# Drop all missing rows that contain only nan values
combined_df = combined_df.dropna(how='all')
# Drop all rows with missing values in Label Column
combined_df = combined_df.dropna(subset=['Label'])

# Fill NAs with zeros
# Change nan values to 0
combined_df = combined_df.fillna(0)

# Duplicate the dataframe
df = combined_df.copy()
df.info()

# ## Decimal To Int

# Convert any floats that contain no decimals into integer datatypes

# ## Seperate Hyphen Values

# Regex to keep only the first value e.g
# -100-100-10 becomes -100,  123-456-1 becomes 123, -10-2 becomes-10, 81-63-63 becomes 81
def seperated_values(x):
    x = str(x)
    match = re.match(r'^(-?\d+).*$', x)
    if match:
        return match.group(1)
    else:
        return x

```

```
# Go through all columns and change separate values into just one value
for column in df.columns:
    df[column] = df[column].apply(seperated_values)
    print('Processing', column)
df['Label'].value_counts()
df['tcp.option_len'].value_counts()
df['ip.ttl'].value_counts()
df['dns.resp.ttl'].value_counts()

# 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.
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"
for index, row in df.iterrows():
    # If the value is a string and matches the regex
    if isinstance(row['tcp.option_len'], str) and re.search(regex, row['tcp.option_len']):
        #print(row['tcp.option_len'])
        df.drop(index, inplace=True) # Drop the value in place by its index

for index, row in df.iterrows():
    # If the value is a string and matches the regex
    if isinstance(row['ip.ttl'], str) and re.search(regex, row['ip.ttl']):
        df.drop(index, inplace=True) # Drop the value in place by its index

for index, row in df.iterrows():
    # If the value is a string and matches the regex
    if isinstance(row['dns.resp.ttl'], str) and re.search(regex, row['dns.resp.ttl']):
        df.drop(index, inplace=True) # Drop the value in place by its index

# ### tcp.option_len
# Use str.match method to apply the regex pattern to the column
mask = df['tcp.option_len'].astype(str).str.match(regex).fillna(False)

# Filter the dataframe
df = df[~mask]
df['tcp.option_len'].value_counts()

# ### dns.resp.ttl
mask = df['dns.resp.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['dns.resp.ttl'].value_counts()

# ### ip.ttl
```

```
mask = df['ip.ttl'].astype(str).str.match(regex).fillna(False)
df = df[~mask]
df['ip.ttl'].value_counts()

# ## Export to CSV
df.to_csv('website_reduced.csv', index=False)
```

G.2 Data Import - Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
import numpy as np

def load_data():
    chunk_size = 1000000
    dtype_opt = {
        'frame.len': 'int64',
        'radiotap.dbm_antsignal': 'int64',
        'radiotap.length': 'int64',
        'radiotap.present.tsft': 'int64',
        'wlan.duration': 'int64',
        'wlan.fc.ds': 'int64',
        'wlan.fc.frag': 'int64',
        'wlan.fc.moredata': 'int64',
        'wlan.fc.protected': 'int64',
        'wlan.fc.pwrmgt': 'int64',
        'wlan.fc.type': 'int64',
        'wlan.fc.retry': 'int64',
        'wlan.fc.subtype': 'int64',
        'wlan_radio.duration': 'int64',
        'wlan_radio.signal_dbm': 'int64',
        'wlan_radio.phy': 'int64',
        'arp': 'object',
        'arp.hw.type': 'object',
        'arp.proto.type': 'int64',
        'arp.hw.size': 'int64',
        'arp.proto.size': 'int64',
        'arp.opcode': 'int64',
        'ip.ttl': 'int64',
        'tcp.analysis': 'int64',
        'tcp.analysis.retransmission': 'int64',
        'tcp.checksum.status': 'int64',
        'tcp.flags.syn': 'int64',
        'tcp.flags.ack': 'int64',
        'tcp.flags.fin': 'int64',
        'tcp.flags.push': 'int64',
        'tcp.flags.reset': 'int64',
        'tcp.option_len': 'int64',
```

```
        'udp.length': 'int64',
        'nbns': 'object',
        'nbss.length': 'int64',
        'ldap': 'object',
        'smb2.cmd': 'int64',
        'dns': 'object',
        'dns.count.answers': 'int64',
        'dns.count.queries': 'int64',
        'dns.resp.ttl': 'int64',
        'http.content_type': 'object',
        'http.request.method': 'object',
        'http.response.code': 'int64',
        'ssh.message_code': 'int64',
        'ssh.packet_length': 'int64'
    }

    # Read the data
    print('Reading X...')
    X = pd.DataFrame()
    for chunk in pd.read_csv('/tf/notebooks/Notebooks/100%/X.csv
    ', chunksize=chunk_size, usecols=dtype_opt.keys(), dtype=
    dtype_opt, low_memory=False):
        X = pd.concat([X, chunk])

    print('Reading y...')
    y = pd.DataFrame()
    for chunk in pd.read_csv('/tf/notebooks/Notebooks/100%/y.csv
    ', chunksize=chunk_size, usecols=['Label'], dtype='object',
    low_memory=False):
        y = pd.concat([y, chunk])

    # Split the data into training and testing sets
    print('Splitting the data...')
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.30, random_state=1234, stratify=y)
    del X, y

    # Scale the data
    print('Scaling the data...')
    scaler = MinMaxScaler()
    scale_cols = ['frame.len',
            'radiotap.dbm_antsignal',
            'radiotap.length',
            'wlan.duration',
            'wlan_radio.duration',
            'wlan_radio.signal_dbm',
            'ip.ttl',
            'udp.length',
            'nbss.length',
            'dns.count.answers',
            'dns.count.queries',
            'dns.resp.ttl',
            'ssh.packet_length']

```

```
    X_train[scale_cols] = scaler.fit_transform(X_train[scale_cols])
    X_test[scale_cols] = scaler.transform(X_test[scale_cols])

    # Encode the labels
    print('Encoding X...')
    cols_to_encode = [col for col in X_train.columns if col not in scale_cols]
    X_all = pd.concat([X_train, X_test], axis=0)
    X_all_ohe = pd.get_dummies(X_all, columns=cols_to_encode, drop_first=True, dtype=np.uint8)
    X_train_ohe = X_all_ohe[:len(X_train)]
    X_test_ohe = X_all_ohe[len(X_train):]
    del X_all
    del X_all_ohe

    print('Label Encoding y...')
    le = LabelEncoder()
    y_train_encoded = le.fit_transform(y_train.values.ravel())
    y_test_encoded = le.transform(y_test.values.ravel())
    del y_train, y_test

    label_mapping = dict(zip(le.classes_, range(len(le.classes_))))
    print(label_mapping)

    return X_train_ohe, y_train_encoded, X_test_ohe, y_test_encoded
```

G.3 Data Import 2 - Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
import numpy as np


def load_data():
    chunk_size = 1000000
    dtype_opt = {
        'frame.len': 'int64',
        'radiotap.dbm_antsignal': 'int64',
        'radiotap.length': 'int64',
        'radiotap.present.tsft': 'int64',
        'wlan.duration': 'int64',
        'wlan.fc.ds': 'int64',
        'wlan.fc.frag': 'int64',
```

```
        'wlan.fc.moredata': 'int64',
        'wlan.fc.protected': 'int64',
        'wlan.fc.pwrmgt': 'int64',
        'wlan.fc.type': 'int64',
        'wlan.fc.retry': 'int64',
        'wlan.fc.subtype': 'int64',
        'wlan_radio.duration': 'int64',
        'wlan_radio.signal_dbm': 'int64',
        'wlan_radio.phy': 'int64',
        'arp': 'object',
        'arp.hw.type': 'object',
        'arp.proto.type': 'int64',
        'arp.hw.size': 'int64',
        'arp.proto.size': 'int64',
        'arp.opcode': 'int64',
        'ip.ttl': 'int64',
        'tcp.analysis': 'int64',
        'tcp.analysis.retransmission': 'int64',
        'tcp.checksum.status': 'int64',
        'tcp.flags.syn': 'int64',
        'tcp.flags.ack': 'int64',
        'tcp.flags.fin': 'int64',
        'tcp.flags.push': 'int64',
        'tcp.flags.reset': 'int64',
        'tcp.option_len': 'int64',
        'udp.length': 'int64',
        'nbns': 'object',
        'nbss.length': 'int64',
        'ldap': 'object',
        'smb2.cmd': 'int64',
        'dns': 'object',
        'dns.count.answers': 'int64',
        'dns.count.queries': 'int64',
        'dns.resp.ttl': 'int64',
        'http.content_type': 'object',
        'http.request.method': 'object',
        'http.response.code': 'int64',
        'ssh.message_code': 'int64',
        'ssh.packet_length': 'int64'
    }

    # Read the data
    print('Reading X...')
    X = pd.DataFrame()
    for chunk in pd.read_csv('/Users/u2054584/Documents/ml-project/Datasets/combined/100%/X.csv', chunksize=chunk_size, usecols=dtype_opt.keys(), dtype=dtype_opt, low_memory=False):
        X = pd.concat([X, chunk])

    print('Reading y...')
    y = pd.DataFrame()
    for chunk in pd.read_csv('/Users/u2054584/Documents/ml-project/Datasets/combined/100%/y.csv', chunksize=chunk_size, usecols=['Label'], dtype='object', low_memory=False):
```

```
        y = pd.concat([y, chunk])

    # Split the data into training and testing sets
    print('Splitting the data...')
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1234, stratify=y)
    del X, y

    # Scale the data
    print('Scaling the data...')
    scaler = MinMaxScaler()
    scale_cols = [
        'frame.len',
        'radiotap.dbm_antsignal',
        'radiotap.length',
        'wlan.duration',
        'wlan_radio.duration',
        'wlan_radio.signal_dbm',
        'ip.ttl',
        'udp.length',
        'nbss.length',
        'dns.count.answers',
        'dns.count.queries',
        'dns.resp.ttl',
        'ssh.packet_length']

    X_train[scale_cols] = scaler.fit_transform(X_train[scale_cols])
    X_test[scale_cols] = scaler.transform(X_test[scale_cols])

    # Encode the labels
    print('Encoding X...')
    cols_to_encode = [col for col in X_train.columns if col not in scale_cols]
    X_all = pd.concat([X_train, X_test], axis=0)
    X_all_ohe = pd.get_dummies(X_all, columns=cols_to_encode, drop_first=True, dtype=np.uint8)
    X_train_ohe = X_all_ohe[:len(X_train)]
    X_test_ohe = X_all_ohe[len(X_train):]
    del X_all
    del X_all_ohe

    print('Label Encoding y...')
    le = LabelEncoder()
    y_train_encoded = le.fit_transform(y_train.values.ravel())
    y_test_encoded = le.transform(y_test.values.ravel())
    y_train_ohe = to_categorical(y_train_encoded)
    y_test_ohe = to_categorical(y_test_encoded)
    del y_train, y_test

    return X_train_ohe, y_train_ohe, y_train_encoded, X_test_ohe, y_test_ohe, y_test_encoded
```

G.4 RF Model 1 - Code

```
# Create the model

start_time = time.time()

# Set up stratified k-fold CV
n_splits = 10
skf = StratifiedKFold(n_splits=n_splits, shuffle=True,
    random_state=1234)

auc_list = []
f1_list = []
precision_list = []
recall_list = []
accuracy_list = []

# Iterate over each fold
for fold_index, (train_index, val_index) in enumerate(skf.split(
    X_train_ohe, y_train_encoded)):

    fold_time = time.time()
    print(f'Fold {fold_index+1}')

    # Split the data into training and validation sets for this
    fold
    X_train_fold, X_val_fold = X_train_ohe.iloc[train_index],
    X_train_ohe.iloc[val_index]
    y_train_fold, y_val_fold = y_train_encoded[train_index],
    y_train_encoded[val_index]

    rf = RandomForestClassifier(random_state=1234, n_jobs=-1,
    verbose=1)
    rf.fit(X_train_fold, y_train_fold)

    # Make predictions on the validation set
    y_pred_fold = rf.predict(X_val_fold)
    y_pred_proba = rf.predict_proba(X_val_fold)

    # Compute evaluation metrics for this fold
    auc = roc_auc_score(y_val_fold, y_pred_proba, multi_class='
    ovr', average='weighted')
    f1 = f1_score(y_val_fold, y_pred_fold, average='weighted')
    precision = precision_score(y_val_fold, y_pred_fold, average
    ='weighted')
    recall = recall_score(y_val_fold, y_pred_fold, average='
    weighted')
    accuracy = accuracy_score(y_val_fold, y_pred_fold)

    # Append the evaluation metrics to the lists
    auc_list.append(auc)
    f1_list.append(f1)
    precision_list.append(precision)
```

```
    recall_list.append(recall)
    accuracy_list.append(accuracy)

    # Print the evaluation metrics for this fold
    print(f"Fold {fold_index+1}: AUC = {auc:.4f}, F1 = {f1:.4f},
     Precision = {precision:.4f}, Recall = {recall:.4f}, Accuracy
     = {accuracy:.4f}")

    print("Fold Time: ", time.time() - fold_time, " seconds")

# Compute the average of the evaluation metrics across all folds
auc_mean = np.mean(auc_list)
f1_mean = np.mean(f1_list)
precision_mean = np.mean(precision_list)
recall_mean = np.mean(recall_list)
accuracy_mean = np.mean(accuracy_list)

# Print the average of the evaluation metrics across all folds
print(f"\nMean AUC = {auc_mean:.4f}, Mean F1 = {f1_mean:.4f},
    Mean Precision = {precision_mean:.4f}, Mean Recall = {
    recall_mean:.4f}, Mean Accuracy = {accuracy_mean:.4f}")

end_time = time.time()
print('Time for Training', end_time - start_time,  ' seconds')

print('\n Creating Final Model with Full Training Data....')

rf = RandomForestClassifier(random_state=1234, n_jobs=-1,
    verbose=2)
rf.fit(X_train_ohe, y_train_encoded)

# Evaluate the model
print('Evaluating...')

rf_y_pred = rf.predict(X_test_ohe)
rf_pred_proba = rf.predict_proba(X_test_ohe)

print('Weighted AUC: ', roc_auc_score(y_test_encoded, rf_y_pred,
     multi_class='ovr', average='weighted'))
print('Weighted Precision: ', precision_score(y_test_encoded,
    rf_y_pred, average='weighted'))
print('Weighted Recall: ', recall_score(y_test_encoded,
    rf_y_pred, average='weighted'))
print('Weighted F1: ', f1_score(y_test_encoded, rf_y_pred,
    average='weighted'))
print('Accuracy: ', accuracy_score(y_test_encoded, rf_y_pred))


report = classification_report(y_test_encoded, rf_y_pred)
print(report)

confusion = confusion_matrix(y_test_encoded, rf_y_pred)
print('Confusion Matrix\n')
print(confusion)
```

```
# Plot the confusion matrix

labels = ['Botnet', 'Malware', 'Normal', 'SQL Injection', 'SSDP'
    , 'SSH', 'Website Spoofing' ]
plt.figure(figsize=(10,10))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',
    xticklabels=labels, yticklabels=labels)
plt.title('RF - Stock 100% - Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.savefig('cm.png')

plt.figure(figsize=(15,15))
feat_importance = pd.Series(rf.feature_importances_, index=
    X_train_ohe.columns)
feat_importance.nlargest(20).plot(kind='barh')
plt.savefig('feature_imp.png')

filename = 'rf.joblib'
joblib.dump(rf, filename)
```

G.5 XGBoost Model 10 - Code

```
print('Training...')

start_time = time.time()
xgb = XGBClassifier(tree_method='gpu_hist', gpu_id=0,
                    eval_metric='merror', early_stopping_rounds
    =10,
                    subsample=0.9, n_estimators=200,
    min_child_weight=3,
                    max_depth=9, learning_rate=0.3, gamma=0,
    colsample_bytree=0.7, verbose=1)

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state
    =1234)

aucs = []
f1s = []
precisions = []
recalls = []
accuracies = []

for train_index, test_index in skf.split(X_train_ohe,
    y_train_encoded):
    start_time = time.time()

    print('Training Fold....')

```

```python
    X_train_fold, X_test_fold = X_train_ohe.iloc[train_index], X_train_ohe.iloc[test_index]
    y_train_fold, y_test_fold = y_train_encoded[train_index], y_train_encoded[test_index]

    xgb.fit(X_train_fold, y_train_fold, eval_set=[(X_test_fold, y_test_fold)], verbose=False)

    y_pred = xgb.predict(X_test_fold)
    y_pred_proba = xgb.predict_proba(X_test_fold)

    # Calculate and store the weighted metrics
    aucs.append(roc_auc_score(y_test_fold, y_pred_proba, multi_class='ovr', average='weighted'))
    f1s.append(f1_score(y_test_fold, y_pred, average='weighted'))
    precisions.append(precision_score(y_test_fold, y_pred, average='weighted'))
    recalls.append(recall_score(y_test_fold, y_pred, average='weighted'))
    accuracies.append(accuracy_score(y_test_fold, y_pred))

    print('Fold Metrics: ', "AUC: ", aucs[-1], "F1-score: ", f1s[-1], "Precision: ", precisions[-1], "Recall: ", recalls[-1], "Accuracy: ", accuracies[-1], "\n")
    elapsed_time = time.time() - start_time
    print(f'Time taken for fold: {elapsed_time:.2f} seconds')

# Calculate the average metrics
avg_auc = np.mean(aucs)
avg_f1 = np.mean(f1s)
avg_precision = np.mean(precisions)
avg_recall = np.mean(recalls)
avg_accuracy = np.mean(accuracies)

print("Average AUC:", avg_auc)
print("Average F1-score:", avg_f1)
print("Average Precision:", avg_precision)
print("Average Recall:", avg_recall)
print("Average Accuracy:", avg_accuracy)
elapsed_time = time.time() - start_time
print(f'Time taken for CV training: {elapsed_time:.2f} seconds')


# Train a final model on the entire training dataset and then evaluate its performance on the test set:
print('Training on entire training dataset...')
start_time = time.time()
eval_set = [(X_test_ohe, y_test_encoded)]
xgb.fit(X_train_ohe, y_train_encoded, eval_set=eval_set, verbose=True)
elapsed_time = time.time() - start_time
print(f'Time taken for final evaluation training: {elapsed_time:.2f} seconds')
```

```
print('Evaluating on test set...')

y_pred = xgb.predict(X_test_ohe)
predictions = [round(value) for value in y_pred]

# evaluate predictions

print('Test ROC AUC: ', roc_auc_score(y_test_encoded, xgb.
    predict_proba(X_test_ohe), multi_class='ovr'))
print('Test Precision: ', precision_score(y_test_encoded, y_pred
    , average='weighted'))
print('Test Recall: ', recall_score(y_test_encoded, y_pred,
    average='weighted'))
print('Test F1: ', f1_score(y_test_encoded, y_pred, average='
    weighted'))
print("Test Accuracy: ", accuracy_score(y_test_encoded, y_pred))

report = classification_report(y_test_encoded, y_pred)
print(report)

confusion = confusion_matrix(y_test_encoded, y_pred)
print('Confusion Matrix\n')
print(confusion)

# Plot the confusion matrix

labels = ['Botnet', 'Malware', 'Normal', 'SQL Injection', 'SSDP'
    , 'SSH', 'Website Spoofing' ]
plt.figure(figsize=(10,10))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',
    xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.savefig('cm.png')

filename = 'xgb.joblib'
joblib.dump(xgb, filename)
```

G.6 MLP Model 4 - Code

```
#  Train the model
print('Training...')

# Stratified K-Fold CV

# Define the number of folds
k_folds = 10

```

```python
skf = StratifiedKFold(n_splits=k_folds, shuffle=True,
    random_state=1234)

# Define lists to store the evaluation metrics for each fold
auc_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
accuracy_scores = []

# Record how long it took to train
start_time = time.time()

for fold_index, (train_index, val_index) in enumerate(skf.split(
    X_train_ohe, y_train_encoded)):
    plt.figure()
    print('\n----------Fold', fold_index+1, '----------')
    X_train_fold, X_val_fold = X_train_ohe.iloc[train_index],
    X_train_ohe.iloc[val_index]
    y_train_fold, y_val_fold = y_train_ohe[train_index],
    y_train_ohe[val_index]

    # Reset column names of X_train_ohe
    X_train_fold.columns = X_train_ohe.columns

    # Create the MLP
    model = Sequential()
    input_shape = (X_train_fold.shape[1],)
    early_stopping = EarlyStopping(monitor='val_loss', patience
    =2)

    model.add(Dense(units=128, activation='relu',
    kernel_initializer=he_uniform(), input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(units=64, activation='relu',
    kernel_initializer=he_uniform()))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(units=32, activation='relu',
    kernel_initializer=he_uniform()))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(units=y_train_ohe.shape[1], activation='
    softmax'))

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.001), loss='
    categorical_crossentropy', metrics=[AUC()])

    # Train the model
```

```
    start_time = time.time()
    history = model.fit(X_train_fold, y_train_fold,
    validation_data=(X_val_fold, y_val_fold), callbacks=[
    early_stopping], batch_size=200, epochs=20)
    end_time = time.time()

    print('Time to train fold:', end_time - start_time)

    # Predict on the test set and calculate metrics
    y_pred = np.argmax(model.predict(X_test_ohe), axis=1)
    y_pred_ohe = to_categorical(y_pred)

    auc = roc_auc_score(y_test_ohe, model.predict(X_test_ohe),
    multi_class='ovr')
    accuracy = accuracy_score(y_test_encoded, y_pred)
    precision = precision_score(y_test_encoded, y_pred, average=
    'weighted')
    recall = recall_score(y_test_encoded, y_pred, average='
    weighted')
    f1 = f1_score(y_test_encoded, y_pred, average='weighted')

    print("Fold %d - AUC: %.4f, Accuracy: %.4f, Precision: %.4f,
     Recall: %.4f, F1: %.4f" % (fold_index+1, auc, accuracy,
    precision, recall, f1))

    # Save the metrics for this fold
    auc_scores.append(auc)
    accuracy_scores.append(accuracy)
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)

    train_loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(train_loss) + 1)

    plt.plot(epochs, train_loss, 'r', label='Training Loss')
    plt.plot(epochs, val_loss, 'b', label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Num of Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig('MLP_Fold_' + str(fold_index+1) + '_Loss.png')

    del history
    K.clear_session() # Clear the keras session

# Calculate the average metrics for all folds
print("Average Metrics - AUC: %.4f, Accuracy: %.4f, Precision:
    %.4f, Recall: %.4f, F1: %.4f" % (np.mean(auc_scores), np.mean
    (accuracy_scores), np.mean(precision_scores), np.mean(
    recall_scores), np.mean(f1_scores)))

end_time = time.time()
```

```
# Print the time it took to train
print('Time for CV:', end_time - start_time, ' seconds')


print('\n Creating Final Model with Full Training Data....')

# Train the model on the entire training set
del model
plt.figure()
model = Sequential()
input_shape = (X_train_fold.shape[1],)
early_stopping = EarlyStopping(monitor='val_loss', patience=2)

model.add(Dense(units=128, activation='relu', kernel_initializer
    =he_uniform(), input_shape=input_shape))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(units=64, activation='relu', kernel_initializer=
    he_uniform()))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(units=32, activation='relu', kernel_initializer=
    he_uniform()))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(units=y_train_ohe.shape[1], activation='softmax'
    ))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='
    categorical_crossentropy', metrics=[AUC()])

# Train the model
history = model.fit(X_train_ohe, y_train_ohe, batch_size=200,
    callbacks=[early_stopping], epochs=20)

plt.plot(history.history['loss'])
plt.title('Training Loss vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.savefig('MLP_Training_Loss.png')

print('\n Evaluating on test set...')

test_loss, test_auc = model.evaluate(X_test_ohe, y_test_ohe)

y_pred = model.predict(X_test_ohe)
y_pred_classes = np.argmax(y_pred, axis=1)
y_pred = np.argmax(y_pred, axis=1)
y_test_encoded = np.argmax(y_test_ohe, axis=1)
```

```

# Save the trained model
print('Saving Model...')
model.save('MLP.h5')

print('Weighted Test Accuracy: ', accuracy_score(y_test_encoded,
    y_pred))
print('Weighted Test Precision: ', precision_score(
    y_test_encoded, y_pred, average='weighted'))
print('Weighted Test Recall: ', recall_score(y_test_encoded,
    y_pred, average='weighted'))
print('Weighted Test AUC: ', roc_auc_score(y_test_ohe, y_pred,
    multi_class='ovr'))
print('Weighted Test F1: ', f1_score(y_test_encoded, y_pred,
    average='weighted'))

print('Confusion Matrix: \n', confusion_matrix(y_test_encoded,
    y_pred))
print('\nClassification Report: \n', classification_report(
    y_test_encoded, y_pred))
```