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

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

Abstract

Abbreviations

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

Contents

List of Figures

List of Tables

1 Introduction

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

1.0.1 Wireless Networks And Attacks

The 802.11 standards have advanced and improved since their inception in 1997 in terms of security, however, despite this, Wi-Fi networks are still vulnerable to well-known attacks such as de-authentication attacks to disconnect all devices from a network, leading to more advanced attacks such as Man-in-the-middle attacks (MITM) or Denial Of Service attacks. The introduction of Protected Management Frames (PMF) in 2009 (IEEE 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 Protected Management Frames (PMF) and Simultaneous Authentication of Equals (SAE) methods of authentication have their shortcomings including being vulnerable to denial-of-service, side-channel, and downgrade attacks (Vanhoef and Ronen 2020).

1.0.2 Intrusion Detection Systems

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

External tools such as Stratosphere IPS (SLIPS) developed by Garcia, 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 into investigating and developing wireless intrusion detection systems using Machine Learning based algorithms utilising supervised, unsupervised and deep learning approaches in both wired and wireless networks. However, research on Intrusion Detection Systems utilising 802.11 and other network protocol features e.g. ARP, TCP & UDP, including application layer features such as HTTP, DNS, SMB etc lacks sufficient research.

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

1.1 Research Questions and Objectives

The objectives for the project are as follows:

- To explore and analyse current literature and academic research utilising ML for intrusion detection systems for IEEE 802.11 networks.
- To examine and identify common machine learning algorithms used for the classification in the context of network attacks.
- To compare the performance of various machine learning models in detecting application layer-based network attacks on an 802.11 dataset, proving a recommendation for a proposed Wireless Intrusion Detection System (WIDS)

- How does combining 802.11 specific and non-802.11 with application layer features affect the detection of application layer attacks?

2 Literature Review

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

2.1 Intrusion Detection Systems

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

2.2 Datasets

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

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

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

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

2.3 Detecting Network Attacks

Application Layer Attacks

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

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

5G Attacks

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

Attack Classifications

10.1007/978-3-030-98457-1˙1 utilised the AWID dataset to predict one of four attacks using the KNN classifier, the paper presented strong results for the "ARP" attack type, achieving the best accuracy with recall. The paper highlighted the importance of the pre-processing of data, feature selection, and choosing an appropriate classifier and oversampling method. The authors suggested that including additional features in the classification process and testing a more generalized model could improve a model's performance in future research and prevent the curse of dimensionality.

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

2.4 Machine Learning Algorithms

A key area of the work was deciding the machine learning algorithms to use, a combination of classifiers and neural networks were considered in their context of suitability, efficiency and performance. The following algorithms were considered:

2.4.1 Random Forest

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

2.4.2 K-Nearest Neighbor

K-Nearest Neighbor is a non-parametric algorithm that works by finding the k closest neighbours to a given input and classifying it based on the majority class within the k neighbours. During our initial experimentation, we found that KNN took over 24 hours to predict on our test data set. 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 need to have a quick response to detecting these attacks. Whilst KNN has advantages, such as being easy to implement and interpret, given the limitations in hardware, we ultimately prioritised speed and accuracy for this work.

2.4.3 XGBoost

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

2.4.4 Neural Networks

Multi-Layer Perceptron

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

2.5 Summary

Based on the literature review and research on the AWID3 dataset and wireless network attack classification, it appears that detecting application layer wireless network attacks using machine learning remains an under-researched area. In their previous work, Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. 2022 showed that combining 802.11 and non-802.11 features achieved high accuracy and AUC, without using application layer features such as DNS, SMB and HTTP etc. However, it remains to be investigated whether combining these application layer features can improve the accuracy of machine learning classifiers in identifying application layer attacks on

802.11 networks. Furthermore, the works fail to classify individually the method of attack, combing 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 802.11 wireless networks.

3 Methodology

3.1 Code Environment

The code for developing the machine learning models were programmed using Python 3.8/9 and Visual Studio Code and Jupyter Notebooks for the IDE. All experiments were conducted on a hardware combination of a 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 32 GB RAM and an Nvidia Tesla M40. Due to the limitations and errors encountered we did not utilise TensorFlow GPU Acceleration for Deep Learning on the M2 Mac Mini.

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

3.2 Libraries

Several libraries were used to develop and implement the machine learning models, including: A selection of common machine learning libraries were 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).

3.3 Feature Selection

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

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

3.3.1 Application Layer Features

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

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

Table 3.1: The selected set of application layer features.

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

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

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

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

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

3.3.2 802.11 Features

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

802.11 Features (16)		
Feature Name	Preprocessing Method	Data Type
radiotap.present.tsft	OHE	int64
wlan.fc.ds	OHE	int64
wlan.fc.frag	OHE	int64
wlan.fc.moredata	OHE	int64
wlan.fc.protected	OHE	int64
wlan.fc.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.

3.3.3 Non-802.11 Features

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

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

Table 3.3: The selected features

3.4 Dataset Manipulation

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

The methodology proposed was as followed:

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

Combing Files

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

After this step, we had 6 large CSV files with the following rows and file size. 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 data-sets, we proceeded to extract the selected features from the 254 features as referenced in Table 3.1, 3.2 and 3.3. Due to the large file sizes, we faced numerous errors and kernel crashes during the importing of the file into Pandas.

Instead of importing all columns, we specified the required features using the 'use_cols' parameter along with the 'chunksize' parameter to read the file in smaller chunks to save memory and eventually combined together, forming one data frame. This saw a reduction in import time and lower memory consumption.

Data Cleaning

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

Upon analysis, we noticed frequent occurrences of hyphened values 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, 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. We followed a similar approach to ibid., 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.

Following on, invalid values were observed, we noticed the presence of value containing month such as: Oct-26, Oct-18, Feb-10 etc. We determined this 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 the invalid values would be dropped from the data. A similar RegEX expression was written to filter out these values from the following columns: *'tcp.option_len', 'dns.resp.ttl', 'ip.ttl', 'smb2.cmd'.* The full code for this section can be found in Appendix A.2.

Individual Datasets

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

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

Table 3.5: Data After Cleaning and Processing

Combining Datasets

Finally, utilising the same bash script (A.1) we combined the six reduced CSV files into one large single dataframe which we exported to a CSV file. The resulting file was 2.67GB in size and contained approximately 21,348,614 rows.

3.5 Pre-Processing Methods

3.5.1 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.

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

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

3.5.2 Normalisation

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

3.6 Data Balancing

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

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

Table 3.6: Data Model Split into Train and Test Sets

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

3.7 Cross Validation

Due to the imbalanced nature of the datasets, stratified k-fold cross validation with a k value of 10 was used, similar to the works carried out by Efstratios Chatzoglou, Georgios Kambourakis, Smiliotopoulos, et al. The training set will be split into 10 folds, the model is then trained on all folds, except one called the validation set. The model is then tested on the validation set for its performance metrics and recorded. This is then repeated for all 10 folds, so each fold is used as a test set. The results are then average to obtain a good representation of the model's 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, we train the model using the full training set and evaluate it based on the test set to obtain a final measure of performance, before finally saving the model.

3.8 Machine Learning Algorithms

3.9 Evaluation Metrics

A key area of the work was deciding the specific metrics use to evaluate the performance of the models. Metrics are vital to determine if models were under or over-fitting on our data and helps to provide context into steps and modifications needed to improve the performances of our models. As a multi-class classification problem, we concerned on primarily two main metrics of evaluation:

AUC-ROC

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

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

As our problem is multi-class, the AUC will be calculated by computing the one-vs-all metric for each class separately i.e, calculated for each class individually, treating all samples for that class as positive and all other 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. Recall is the fraction of correctly predicted positive instances out of the total actual positive instances.

The F1-score was chosen due to its representation in an imbalanced dataset; as it considers both precision and recall. Accuracy can be a misleading metric

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 our multi-class classification problem a slightly different approach must be taken. In particular, there are three main methods:

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

Therefore, the weighted averaging method was chosen, leading to robust scores that takes into account both the number of samples within the class and its performance. It was observed that most previous works fails 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 of detailing the metrics for Precision, Recall, Accuracy and F1 across each class.

Confusion Matrix

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

4 Experiments

4.1 Initial Modelling

In order to speed up initial training and testing for each machine learning algorithm, a multitude of subsets of the original combined data were created using sklearn's train_test_split to create a stratified split resulting in reduced data sets. Varying levels of data splits were created, including a 50%, 60% and 80% data split from the original 12 million rows of data as seen in Table 3.6. Furthermore, due to the limitations in hardware and training time, the majority of models were optimised on a trial and error methodology to find the optimal parameters, with some instances where GridSearchCV was used. We did not use 10 fold Stratified Cross Validation during initial experiments, this was justified under the pretense that the best found parameters would undergo cross validation training at a later stage.

4.2 Classifiers

Classification on the combined set of features was conducted against three classifier models, Random Forest (RF), XGBoost and K-Nearest Neighbour (KNN).

4.2.1 Random Forest (RF)

Basically existing research doesn't have any models or research using a random forest ensemble classifier, only decision trees in (E. Chatzoglou, G. Kambourakis, and C. Kolias 2021)

Table 4.1: Parameters for Random Forest Classifier

Parameter	**Description**	**Value**
n_estimators	The number of trees in the forest	100
criterion	Function to measure the quality of a split.	gini
max_depth	Maximum depth of the tree.	None
min_samples_split	Minimum samples required to split an internal node.	2
min_samples_leaf	Minimum samples required to be at a leaf node.	1
max_features	Maximum features to consider when splitting.	auto
bootstrap	To bootstrap samples when constructing trees	True
class_weight	Weights associated with classes	None
random_state	The random seed	1234

Table 4.2: RF Model Metrics

Model	**Data Subset**	**Accuracy**	**Precision**	**Recall**	**F1**
Base	80%	0.997	0.955	0.882	0.916
Base	100%	0.997	0.997	0.997	0.997

Table 4.3: RF Model v2

Device	**Model**	**Data Size**	**AUC**	**Precision**	**Recall**	**Accuracy**	**F1**
GPU	Optimised	100%	99.99	99.66	99.67	99.67	99.66

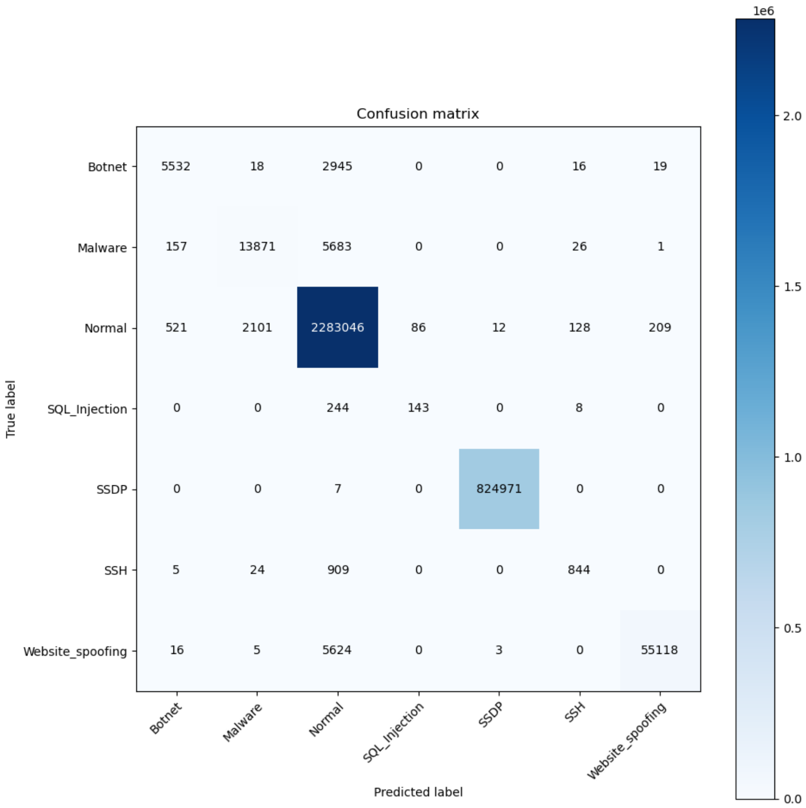

Figure 4.1: Confusion Matrix

Confusion Matrix

4.2.2 XGBoost

Table 4.4: XGB Model Metrics

Model	Device	Dataset	AUC	Accuracy	Precision	Recall	F1
Base	M2	80%	?	99.7	95.5	88.2	91.5
Base	GPU	60%	?	99.6	95.0	88.4	91.4
Base	GPU	80%	1.00	99.6	94.9	87.8	91.0
Base	GPU	100%	?	99.7	99.7	99.7	99.6
Optimised	GPU	80%	1.00	99.6	99.6	99.6	99.6
Optimised	GPU	100%	1.00	99.7	99.6	99.7	99.6

4.3 Neural Networks

4.3.1 Multi-Layer Perceptron (MLP)

Table 4.5 specifies the parameter values for a multi-layered feed forward neural network model, consisting of one input layer (128 neurons) and one hidden layer (64 neurons) using the ReLu activator function. The output layer has a subsequent 7 neurons corresponding to the 7 different output classes, using a soft-max activation function to produce the class probabilities. See Appendix E.1 for the full code.

Table 4.5: MLP v1 Specifications

Parameter	**Value**
Activator	Relu
Output Activator	Softmax
Initialiser	Default
Optimiser	Adam
Momentum	N/A
Early Stopping	N/A
Dropout	N/A
Learning Rate	Default
Loss	Categorical Crossentropy
Batch Norm	N/A
Hidden Layers	2
Nodes per Layer	128, 64
Batch Size	32
Epochs	10

Table 4.6: MLP v1 Classification Report

Class	Precision	Recall	F1-Score	Support
Botnet	0.65	0.53	0.58	8530
Malware	0.83	0.68	0.75	19738
Normal	0.99	1.00	0.99	2286103
SQL Injection	0.98	**0.23**	0.37	395
SSDP	1.00	1.00	1.00	824978
SSH	**0.58**	0.38	**0.46**	1782
Website Spoofing	0.92	0.92	0.92	60766
Accuracy			0.99	3202292
Macro Avg	0.85	0.68	0.72	3202292
Weighted Avg	0.99	0.99	0.99	3202292

After tuning, we proceeded to create an additional model with additional layers

Table 4.7: MLP v2 Specifications

Parameter	**Value**
Activator	Relu
Output Activator	Softmax
Initialiser	he_uniform
Optimiser	Adam
Momentum	N/A
Early Stopping	N/A
Dropout	0.2
Learning Rate	0.001
Loss	Categorical Crossentropy
Batch Norm	Yes
Hidden Layers	3
Nodes per Layer	128, 64, 32
Batch Size	180
Epochs	15

Table 4.8: MLP v3 Specifications

Parameter	**Value**
Activator	Relu
Output Activator	Softmax
Initialiser	he_uniform
Optimiser	Adam
Momentum	N/A
Early Stopping	True (2 rounds)
Dropout	0.2
Learning Rate	0.001
Loss	Categorical Crossentropy
Batch Norm	Yes
Hidden Layers	3
Nodes per Layer	128, 64, 32
Batch Size	200
Epochs	20

Parameter	Model 1	Model 2	Model 3	Model 4	Model 5
Activator:	ReLU	ReLU	ReLU	ReLU	ReLU
Output Activator:	Softmax	Softmax	Softmax	Softmax	Softmax
Initialiser:	Default	he uniform	he_uniform	he_uniform	he uniform
Optimiser:	Adam	Adam	Adam	Adam	SGD
Momentum:	N/A	N/A	N/A	N/A	0.9
Early Stopping:	N/A	N/A	2 Rounds	2 Rounds	N/A
Dropout:	N/A	0.2	0.2	0.2	0.25, 0.2
Learning Rate:	Default	0.001	0.001	0.001	0.01
Loss:	CC	CC	CC	SCC	CC
Batch Norm:	N/A	Yes	Yes	Yes	Yes
Hidden Layers:	2	3	3	3	4
Nodes per Layer:	128, 64	128, 64, 32	128, 64, 32	128, 64, 32	100, 80, 60, 40, 20
Batch Size:	32	180	200	200	180
Epochs:	10	15	20	20	15

5 Analysis Of Results

Bibliography

Chatzoglou, E., G. Kambourakis, and C. Kolias (2021). "Empirical evaluation of attacks against IEEE 802.11 enterprise networks: The AWID3 dataset". In: *IEEE Access* 9, pp. 34188–34205. DOI: `10.1109/ACCESS.2021.3061609`.

Chatzoglou, Efstratios, Georgios Kambourakis, Constantinos Kolias, et al. (2022). "Pick Quality Over Quantity: Expert Feature Selection and Data Preprocessing for 802.11 Intrusion Detection Systems". In: *IEEE Access* 10, pp. 64761–64784. DOI: `10.1109/ACCESS.2022.3183597`.

Chatzoglou, Efstratios, Georgios Kambourakis, Christos Smiliotopoulos, et al. (2022). "Best of Both Worlds: Detecting Application Layer Attacks through 802.11 and Non-802.11 Features". In: *Sensors* 22.15. ISSN: 1424-8220. DOI: `10.3390/s22155633`. URL: `https://www.mdpi.com/1424-8220/22/15/5633`.

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: `10.1145/2939672.2939785`. URL: `http://doi.acm.org/10.1145/2939672.2939785`.

d'Otreppe, Thomas (Nov. 2011). *OpenWIPS-ng*. OpenWIPS-ng. URL: `https://openwips-ng.org` (visited on 10/01/2022).

Dalal, Neil et al. (2021). "A Wireless Intrusion Detection System for 802.11 WPA3 Networks". In: *CoRR* abs/2110.04259. arXiv: `2110.04259`. URL: `https://arxiv.org/abs/2110.04259`.

Distribution, Anaconda Software (2016). *Anacoda*. Anaconda Software Distribution. URL: `https://anaconda.com` (visited on 12/15/2022).

Fawcett, Tom (2006). "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8. ROC Analysis in Pattern Recognition, pp. 861–874. ISSN: 0167-8655. DOI: `https://doi.org/10.1016/j.patrec.2005.10.010`. URL: `https://www.sciencedirect.com/science/article/pii/S016786550500303X`.

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).

Grandini, Margherita, Enrico Bagli, and Giorgio Visani (2020). *Metrics for Multi-Class Classification: an Overview*. arXiv: 2008.05756 [stat.ML].

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.

IEEE (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.

Kismet (July 2002). *Kismet*. URL: https://www.kismetwireless.net (visited on 09/30/2022).

Koço, Sokol and Cécile Capponi (13–15 Nov 2013). "On multi-class classification through the minimization of the confusion matrix norm". In: *Proceedings of the 5th Asian Conference on Machine Learning*. Ed. by Cheng Soon Ong and Tu Bao Ho. Vol. 29. Proceedings of Machine Learning Research. Australian National University, Canberra, Australia: PMLR, pp. 277–292. URL: https://proceedings.mlr.press/v29/Koco13.html.

Kolias, Constantinos et al. (2016). "Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset". In: *IEEE Communications Surveys & Tutorials* 18.1, pp. 184–208.

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

Mughaid, Ala et al. (Sept. 2022). "Improved dropping attacks detecting system in 5g networks using machine learning and deep learning approaches". In: *Multimedia Tools and Applications*. ISSN: 1573-7721. DOI: 10.1007/s11042-022-13914-9. URL: https://doi.org/10.1007/s11042-022-13914-9.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Roundy, Jacob (May 2021). *IoT Security: IoT Device Security Challenges & Solutions*. Verizon Enterprise. (Visited on 11/21/2022).

Saskara, Gede Arna Jude et al. (Feb. 2022). "Performance of Kismet Wireless Intrusion Detection System on Raspberry Pi". In: EAI. DOI: 10.4108/eai.27-11-2021.2315535.

Satam, Pratik and Salim Hariri (2021). "WIDS: An Anomaly Based Intrusion Detection System for Wi-Fi (IEEE 802.11) Protocol". In: *IEEE Transactions on Network and Service Management* 18.1, pp. 1077–1091. DOI: 10.1109/TNSM.2020.3036138.

Vanhoef, Mathy and Frank Piessens (2017). "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, pp. 1313–1328. ISBN: 9781450349468. DOI: 10.1145/3133956.3134027. URL: https://doi.org/10.1145/3133956.3134027.

Vanhoef, Mathy and Eyal Ronen (2020). "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd". In: *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 517–533. DOI: 10.1109/SP40000.2020.00031.

WPA3 Specification Version 3.1 (Nov. 2022). Tech. rep. Wi-Fi Alliance. URL: https://www.wi-fi.org/file/wpa3-specification (visited on 10/03/2022).

A Data-set Manipulation

A.1 CSV Combiner Script

```
#!/bin/bash

# Input Directory
input_dir="../Malware"

cd "$input_dir"

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

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

# Print a status message
echo "Combining files..."

# Loop through all the files that match the pattern reduced_*.
    csv
for file in $(ls *.csv | sort -V)
do
  # Check if the file exists
  if [ -f "$file" ]; then
    # Print a status message
    echo "Combining $file..."

    # If this is 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

# Print a status message
echo "Done."
```

A.2 Feature Extraction & Reduction

```
# Define the columns you want 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 you want to read in each iteration
batch_size = 1000000

# Initialize an empty dataframe to hold the combined results
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)
```

B Conda Environments

B.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
```

B.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
```

C Data Preprocessing

C.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])
```

C.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):]
```

C.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)
```

C.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)
```

D Classifiers

D.1 Base 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()
```

D.2 XGBoost

D.2.1 Base XGBoost CF

D.2.2 Base XGBoost Classification Report

	precision	recall	f1-score	support
0	0.96	0.75	0.84	13648
1	0.86	0.86	0.86	31581
2	1.00	1.00	1.00	3657765
3	0.94	0.84	0.89	631
4	1.00	1.00	1.00	1319964
5	0.95	0.76	0.84	2852
6	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

E Neural Networks

E.1 MLP NN v1

```
# Create a sequential model
model = Sequential()
input_shape = (X_train_ohe.shape[1],)

# Add layers to the model
model.add(Dense(128, activation='relu', input_shape=input_shape)
    )
model.add(Dense(64, activation='relu'))
model.add(Dense(7, activation='softmax'))

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

# Train the model
model.fit(X_train_ohe, y_train_ohe, epochs=10, batch_size=32,
    validation_data=(X_test_ohe, y_test_ohe))

# Evaluate the model using test data
test_loss, test_acc = model.evaluate(X_test_ohe, y_test_ohe)

print('Test accuracy:', test_acc)
```

E.1.1 MLP Neural Network

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from keras.optimizers import SGD
from keras.initializers import he_uniform
from keras.metrics import AUC

# Define the number of classes
num_classes = 7

# Define the model architecture
model = Sequential()

# Add the input layer
model.add(Dense(100, input_shape=(X_train_ohe.shape[1],),
    activation='relu', kernel_initializer=he_uniform()))

# Add batch normalization
model.add(BatchNormalization())

# Add the first hidden layer
```

```python
model.add(Dense(80, activation='relu', kernel_initializer=
    he_uniform()))
model.add(Dropout(0.25))
model.add(BatchNormalization())

# Add the second hidden layer
model.add(Dense(60, activation='relu', kernel_initializer=
    he_uniform()))
model.add(Dropout(0.2))
model.add(BatchNormalization())

# Add the third hidden layer
model.add(Dense(40, activation='relu', kernel_initializer=
    he_uniform()))
model.add(BatchNormalization())

# Add the fourth hidden layer
model.add(Dense(20, activation='relu', kernel_initializer=
    he_uniform()))
model.add(BatchNormalization())

# Add the output layer
model.add(Dense(num_classes, activation='softmax'))

# Define the optimizer
sgd = SGD(lr=0.01, momentum=0.9)

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

# Train the model
batch_size = 170
epochs = 10
history = model.fit(X_train_ohe, y_train_ohe, batch_size=
    batch_size, epochs=epochs, validation_data=(X_test_ohe,
    y_test_ohe))

# Evaluate the model on your test data
test_loss, test_auc = model.evaluate(X_test_ohe, y_test_ohe)
```