東海大學

資訊工程研究所

碩士論文

指導教授: 楊朝棟博士

共同指導教授: 劉榮春博士

使用 XGBoost 機器學習法進行攻擊檢測與分析並以 ELK Stack 視覺化於網路日誌系統

On Construction of a Network Log System with Cyberattack Detection Using XGBoost on ELK Stack

研究生: 賴慶翰

中華民國一零八年六月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 賴 慶 翰 所提之論文

使用 XGBoost 機器學習法進行攻擊檢測與分析
並以 ELK Stack 視覺化於網路日誌系統

經本委員會審查，符合碩士學位論文標準。

學位考試委員會
召 集 人 許慶賢 簽章

委 員 劉榮春

詹毓偉

賴冠州

指 導 教 授 楊朝棟 簽章

中華民國 108 年 6 月 12 日

摘　要

近年來隨著人工智慧與機器學習技術發展逐漸成熟，越來越多網路攻擊行為以此做為發展方向，企圖將其發展成迴避資訊安全偵察的新手段，這也因此大大增加了資訊安全防護的難度。根據統計 2018 上半年遭洩漏、損毀的數據量相較於 2017 年同期增加了 72%。但令人驚訝的是 2018 上半年獲報的資訊安全事件卻比 2017 年同期少 18%。在這些事件中，多數案例為遭受「進階持續性滲透攻擊」(簡稱 APT)。對於這類型攻擊的防禦方法，可藉由觀察日誌資料並從中分析是否具有異常行為，並以此進行檢測、辨識攻擊事件。本論文將實作 ELK Stack 網路日誌系統 (NetFlow Log)，進行視覺化分析日誌數據，並呈現數種網路攻擊行為特徵，供管理者進一步分析。本論文將導入歷史日誌數據，運用「極限梯度提升」(簡稱 XGBoost) 進行機器學習，以及運用 Keras 進行深度學習，建置一個檢測日誌是否具有攻擊事件之模型。本文的最終目標將透過實驗，在此案例中，尋求最佳的學習模式。而透過實驗證實，在此案例中，XGBoost 機器學習模型對潛在威脅判斷的準確率達 96.01%, 全攻擊數據集可以達到 100% 辨識，優於 RNN、DNN 模型。並且本文將進一步的將實驗結果與網路日誌平台結合，管理者可根據模型判斷結果與 ELK Stack 網路日誌系統互相比對，進行風險評估。本文所運用之日誌資料為持續性數據，目前數據已達 2TB 以上並持續增加中。

關鍵字: 網路安全、機器學習、極限梯度提升、ELK Stack、NetFlow Log

Abstract

With the development of artificial intelligence and machine learning technology, more and more cyber attacks have taken this as a development direction. According to statistics, the amount of data leaked and destroyed in the first half of 2018 increased by 72%compared with the same period in 2017. Of these events, most cases suffer from Advanced Progressive Penetration Attacks (APT). For the defense method of this type of attack, it is possible to detect and identify the attack event by observing the log data and analyzing whether it has abnormal behavior. This paper will be implemented as ELK Stack network log system (NetFlow Log) to visually analyze log data and present several kinds of network attack behavior characteristics for further analysis by managers. This paper will import historical log data, use "extreme gradient enhancement" (XGBoost for machine learning), and use Keras for deep learning to build a model to detect whether the log has an attack event. The ultimate goal of this paper will be to find the best learning model through experiments in this case. Through experiments, it is confirmed that in this case, the XGBoost machine learning model has an accuracy rate of 96.01%for potential threats, and the full attack data set can achieve 100% recognition, which is better than RNN and DNN models. And this article will further combine the experimental results with the network log platform. The administrator can compare the model judgment results with the ELK Stack network log system for risk assessment. The log data used in this paper is continuous data, and the current data has reached more than 2TB and continues to increase.

Keywords: Cyber Security, Machine Learning, ELK Stack, XGBoost, NetFlow Log

致謝詞

這篇論文能夠如此順利完成，首先誠摯的感謝指導教授楊朝棟博士，碩一時進入高效能計算實驗室，教授的悉心指導使學生可以站在計算機科學領域的前沿，一窺現今最新的技術理論與實務研究方向，並給予學生許多不同的訓練與工作，讓學生能於碩士在學期間快速累積專業實力。除了學術研究外，教授在許多做人處事的態度以及應對，更是學生的典範。再次感謝教授在研究所期間的指導，相信這段期間的訓練對我往後的人生都有很大的幫助。

非常感謝口試委員們，許慶賢教授、劉榮春教授、賴冠州教授、詹毓偉教授，百忙之中抽空參加我的論文口試，每位口試委員的指教點評都將使本論文更加完善而嚴謹。感謝高效能計算實驗室的學長姐、學弟們的指教與幫忙。特別謝謝研究所兩年以及大學四年的同儕，有你們的陪伴和協助才能使得本論文的研究更加完整。

最後謝謝我的家人，謝謝父母從小對我的栽培、支持與鼓勵，因為有你們各方面的支持，才能讓我沒有後顧之憂的求學，由衷感謝一路上幫助我和陪伴我的所有人，謝謝你們。

東海大學資訊工程學系 高效能計算實驗室 賴慶翰 一零八年六月

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In October 2015, AlphaGo defeated its professional opponent in the Go game. It was officially published in the famous journal Nature in January 2016 [11]. In this era, artificial intelligence, Machine Learning and Deep Learning came into being. And gradually mature. Artificial intelligence brings enormous potential, such as efficiency and productivity. Companies from all over the world are also exploring business opportunities. Together with big data, they can even calculate and predict unsuccessful results, making decisions more accurate, clear and fast.

However, in today's booming information technology era, information security protection is undoubtedly the most concerned issue for private companies and governments. In 2017, the Encrypting Ransomware Worm, WannaCry spread around the world, and many companies and even national governments were affected [3]. In 2017, Equifax announced that it had suffered a cyber security attack and that nearly 200,000 customers' credit card information was stolen during the attack. In many cases, traditional cyber security protection is clearly not applicable. However, if the traditional cyber security architecture can be combined with artificial intelligence, whether it can have more complete and accurate detection capabilities, provide managers with effective risk assessment information, and even find signs in the early stages of cyber attacks. More targeted anti-blocking or security protection against information data can be performed.

Therefore, in this work, our goal is to use XGBoost for Machine Learning, and then to implement a visualization system for cyber attack behavior to help administrators detect whether historical network log data has cyber attack behavior. The manager assesses the risk of network logs. The specific objectives are as follows:

1. Establish a network log system, store network log data in the server in real time, and perform data preprocessing. Logstash is used to collect log data and convert it to the required data fields.
2. Use Elasticsearch to search for the required information based on the fields, and finally use Kibana to visualize the results of the network log data.
3. Use Anaconda Jupyter Notebook as the development environment to write data preprocessing and XGBoost Machine Learning models.
4. Import historical network log data, use XGBoost Machine Learning model for binary classification prediction, and detect attack behavior.

1.1 Motivation

In recent years, the means of cyber-attacks have been constantly changing. With the development of Machine Learning technology, some illegal users use the technology for cyber-attacks and use Machine Learning aids to help analyze information about social networks. In order to improve the attack efficiency, the specific target of the targeted attack is given according to the information, the attack success rate is improved, or the artificial intelligence is used to attempt the intrusion behavior, the vulnerability is discovered, and the general information security protection is avoided. For example, by Machine Learning to identify words and images, the proof code mechanism for preventing automatic attacks is broken. According to Neustar's International Network Benchmark Index report released in 2018 [2], 82%of cyber security experts said they are worried that attackers will

use artificial intelligence to attack destructive attacks on the network environment, but a large number of experts (87%) believe that artificial intelligence can It plays a huge role in network security and provides great help.

As mentioned above, in the campus network environment, various cyber attacks have appeared and attempts to influence the stability of the campus network environment. From the network logs, can find out there are many unusual network usage scenarios that are trying to pass the campus network security system. However, on today's market, systems with visualized network log data and capable of detecting cyber attacks are subject to considerable charges. However, if the budget is limited, how can I analyze the network log data and visualize the attack behavior in the network log to facilitate the administrator to obtain relevant information for prevention? In this work, the open source platform ELK Stack is used to build a network log system (NetFlow Log). After collecting the network logs, you can find the log data related to the cyber attack behavior in a large amount of data, and obtain preliminary information from it. Observe the analysis. After visualizing the log data, the administrator can use the Machine Learning model to import historical log data into the model for analysis and detection, and perform risk assessment based on the cross-validation analysis of the visual information displayed by the ELK Stack, even if it has not occurred or is uncertain. the manager also have sufficient information to make the right decisions and take precautions to avoid the associated losses in information security.

1.2 Thesis Goal and Contributions

This work builds a network data storage and analysis platform system using ELK Stack. The system is an open source platform that allows for the storage, analysis and visualization of data. In addition, use XGBoost for Machine Learning, build models, and import historical log data to detect the presence of an attack. Here are the original contributions of this work:

1. Using the ELK Stack open source platform, the architecture can fully implement the data processing of the network log, analyze the cyber attack events, and finally visualize, and show how the administrator can use the ELK Stack to observe and monitor the network log data to avoid attacks.

2. This work uses XGBoost for Machine Learning, and Keras for Deep Learning. The attack event detection model can effectively help managers detect possible attack events in historical network logs and conduct risk assessment based on the information provided to make more accurate judgements.

1.3 Thesis Organization

The structure of this work is as follows. Chapter 2 mainly introduces the research background and related work. Chapter 3 provides an overview of the visual analysis of attack events in the ELK Stack log system, using XGBoost for machine learning and use Keras for deep learning, and building models to detect attack events in log data. Next, Chapter 4 details the experimental steps and discussions. Finally, in Chapter 5, the conclusions and future work of this work are provided.

Chapter 2

Background Review and Related Works

In chapter 2, provide the background of this work and several kit information, including Python, ELK Stack, XGBoost and so on. The next section will be discussed in more detail.

2.1 Python and Anaconda3

In this work, the code for Machine Learning is written in Python 3 [10]. According to statistics, Python, C and Java are the three most popular programming languages. Created by Guivo van Rossum, the Python programming language is a widely used object-oriented and interpreted computer programming language. Python has many features that make it one of the most popular programming languages. For example, the code is highly readable. Compared to C or Java, Python's simple syntax makes it easier for beginners to learn, allowing developers to focus more on solving the problem itself than on language logic. Python is also highly portable and embeddable. The Python interpreter can be executed on almost all operating systems, such as Linux, Windows, OS, and does not require modification of Python code. CPython is a Python interpreter written in C. If

developers intentionally avoid over-reliance on a particular system, Python can be executed directly on most systems. Python and C can be embedded in each other. Developers can use C or C++ code. Write programs that are used directly in Python and vice versa.

Anaconda is a free open source suite with built-in kits for Python developers to apply to different types of data analysis, big data processing and Machine Learning. Anaconda also supports different system platforms, and can also be developed using virtual environments. Developers are free to switch between Python 2 and Python 3 to avoid many version conflicts. In addition, Anaconda provides developers with a built-in Spyder compiler and Jupyter notebook environment. This work use Jupyter Notebook as the development environment and use many software packages, such as Scikit-Learn [12], Numpy, Pandas, Matplotlib, Graphviz, etc., to perform data preprocessing and build Machine Learning models.

2.2 Keras

In the field of Deep Learning, CNTK and TensorFlow are widely used in Deep Learning research. Although both have very powerful features, the actual application is more difficult. Therefore, the Deep Learning project for this job will use Keras to build a Deep Learning model.

Keras is an open source neural network library written in Python that can be executed on TensorFlow, CNTK, Theano, and the main developer is Google engineer Francois Chollet. Keras is able to quickly implement deep neural networks. Keras has the following features, which is why this work chose it as a way to achieve Deep Learning.

1. User friendly

   As an API for rapid development of Deep Learning models. The user experience is Keres' first consideration. Kreas provides a consistent and concise API that greatly helps developers reduce their work.

2. Modular

   In Keras, losses, optimizers, initializers, activations, regularizers, etc. are independent modules. You can freely configure these modules at a very low cost and use them to build models.

3. Extensibility

   It's very easy for Keras to add new modules. You need to copy the existing modules to write new classes or functions. The convenience of adding new modules makes Keras very suitable for research work.

Keras has many tools to make it easier to work with images and text. In addition to standard neural networks, Keras also supports Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).

2.3 ELK Stack

ELK Stack refers to the architecture based on three open source softwares Elasticsearch, Logstash and Kibana [7]. These three softwares can be used to form a system for querying, collecting, and analyzing logs. This work can get data from any source and format. Without changing the original system architecture, ELK Sack is built to instantly search and analyze data and ultimately use visual capabilities to present the analyzed data results [4]. NetFlow Log is the automated network log platform built in this work. It is built on top of these three open source software. In addition, ELK Stack not only has three kits, but also many other software packages, such as Filebeat, Xpack, ECE, etc., for developers to apply. The following is a brief introduction to the features of the toolkit used in this work.

Elasticsearch is a JSON-based real-time distributed data search and analysis engine that searches data at extremely fast speeds and scales. At the heart of the ELK Stack, Elasticsarch includes three functions: full-text search, structured

search and analysis. It is highly reliable, easy to manage and scales out. Elasticsarch supports RESTful styles and provides a number of plugins for developers. Can be embedded for use. While the features included in Elasticsarch are not absolutely groundbreaking, they are very simple for first time developers and it can still be satisfied as developers' needs increase.

Logstash is another open source software that provides a framework for data collection and processing. Data is often scattered or concentrated in multiple systems in various forms. Logstash can collect data from multiple sources simultaneously and identify and analyze the collected data. After the conversion split, the data is stored in the appropriate fields, processed and converted to the specified location, whether it is a database or a file.

Kibana is another core open source software for ELK. It has powerful data analysis and visualization capabilities. Kibana can use to search, view, and analyze data from Elasticsearch, and visualize the data in multiple ways. Provide developers or users with easy-to-understand graphics that make it easier for them to understand and get information from the data.

2.4 Machine Learning

From the moment AlphaGo defeated human professional Go players, artificial intelligence has become the most compelling information technology after big data applications. Machine Learning is a branch of artificial intelligence in artificial intelligence. It is a way to achieve artificial intelligence. As the name implies, Machine Learning is the ability to let computers learn automatically from data. Machine Learning is a multidisciplinary, interdisciplinary field that has emerged over the past 25 years, involving statistics, decision theory, probability theory, and many other disciplines. In Machine Learning, we hand over data to a computer. The computer automatically analyzes and identifies the data collected by the computer, then summarizes the rules known to the computer and automatically improves the algorithm through data and learning experience. Optimized

performance standards. Finally, the unknown information is predicted by its algorithm law, which is a Machine Learning algorithm.

At present, Machine Learning has been widely used in our lives, such as weather forecasting, language translation, face recognition [5], license plate recognition, medical diagnosis assistance [6], stock trend analysis, and speech recognition. Machine Learning is divided into supervised learning, semi-supervised learning and unsupervised learning. First, unsupervised learning, and will not give pre-marked training examples. Instead, let the machine summarize the input data to find out the classification that the machine knows. Supervised learning, on the basis of the information given, marks the samples one by one, giving the Machine Learning objectives. Based on the purpose of marking the information we give, the computer can learn a function based on the training dataset, and according to this marked The sample is summarized in the description, when the new data is imported into the Machine Learning model, this unknown information can be predicted. This method is also the main research method used in this work. Compared with the above two learning methods, semi-supervised learning is somewhere in between. In many practical problems, only part of the data can be given to the target, and semi-supervised learning is to reduce the way of giving data goals, thereby improving the efficiency of Machine Learning. There are many ways to learn about Machine Learning. In this chapter, only the part that has been applied in this work is introduced as a research background.

Next, introduce the decision tree. In decision theory, a decision tree consists of decision graphs and possible outcomes that are used to help decision-making achieve program goals [1]. In Machine Learning, a decision tree is a predictive model. Use tree graphics to help computers judge, segment our data, and make decisions based on it. Each node in the tree represents a specific target, each forked path represents a possible feature of data segmentation, and the information gain is obtained from the action of segmenting the data, and the segmentation process is repeated until the leaf node. And this leaf node corresponds to the target from the master node to the leaf node, and has all the feature values on the path. Decision trees are available for forecasting and data analysis. A complete decision

tree typically contains three types of nodes, decision nodes, opportunity nodes, and endpoints. Decision trees have several generation methods, classification tree analysis, regression tree analysis, CART, CHAID. Compared to other data mining algorithms such as K-Means and KNN, the use of decision trees has advantages in the following areas:

1. The decision tree is easy to understand, and the developer can understand the meaning of the model through the decision tree.
2. Other algorithms typically require data to be deleted from redundant or null values compared to decision trees.
3. The decision tree can process data for different data attributes simultaneously.
4. Generate a decision tree by giving features, which will give the corresponding logical representation.
5. Compared with other algorithms, it can get a good solution in a short time.

However, the shortcomings of decision trees are also obvious. If there are large differences in the number of samples in each category, the decision tree will be significantly biased towards features with most samples.

As the most basic component of XGBoost, need to introduce the CART regression tree. It constructs a decision tree based on the characteristics and data of the training, and uses this to determine the prediction result of each piece of data. It uses the gini index to calculate the gain to select the characteristics of the decision tree. The Gini index formula is as follows:

$$Gini(D) = \sum_{k=1}^{K} p_k 1 - p_k \tag{2.1}$$

p_k represents the probability of class classification category (k) in dataset D, the number of categories is indicated by K. The Gini index calculates the gain formula as:

$$Gini(D, A) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2) \tag{2.2}$$

D represents the entire dataset, D_1 and D_2 respectively represent data having feature A in the dataset and data other than A.

Before talking about Gradient Boosting, I will first introduce Boosting. For example, completing a dataser to construct multiple classification models, and the model structure is very simple, such as correct or wrong, call it a weak classifier (weak learner), and then every time after the classification, will be the last time The data weight of the classification error is increased, and then a new model is added to correct the error generated by the existing model, and the classification is performed again until it cannot be further improved, thereby finally obtaining good results of the training data and the test data. Gradient Boosting is a Boosting method that is a Machine Learning technique for regression and classification problems [9]. Gradient Boosting generates prediction models in the form of multiple weak classifiers, and each model is established in the gradient direction of the loss function of the previous model. Simply put, when the loss function is large, the model is more error-prone. If our model can make the loss function continue to drop, then our model will continue to improve. The loss function is reduced in the gradient direction by multiple improvements, and a good model is finally obtained [8]. The specific algorithm is as follows:

Input：Training set T=(x_1,y_1),(x_2,y_2) ... ,(x_n,y_n)
Output：Boosting tree f_M (x)
Procedures:

- Initialization f_0 (x)=0 for m=1,2 ... ,M
- Calculating the residual $r_m i$=y_i-$f_($(m − 1)) (x_i),i=1,2, ... ,n (3)
- Fitting the residual $r_m i$ to learn a regression tree and get T(x:$_m$)
- Update f_m (x)=$f_($(m − 1)) (x)+T(x:$_m$)
- Get the regression boosting tree: f_M (x)= $_($m = 1)^M$ T(x:$_m$)

2.5 XGBoost

In recent years, XGBoost has become one of the most useful algorithmic tools in the Machine Learning field due to the most favorite of many award-winning teams in the Machine Learning competition. It is a gradient supercharger researched and implemented by Chen Tianqi [13]. According to Xqioost founder Tianqi Chen, XGBoost has been used in the production operations of major companies such as Uber, Airbnb, Amazon and Google. XGBoost stands for eXtreme Gradient Boosting. XGBoost is a massively parallel enhanced tree tool designed to provide a scalable, decentralized and lightweight open source software library that helps us improve speed and performance while learning Machine Learning. It provides us with a framework for Gradient Boosting. As described in the previous section, the Gradient Boosting framework takes advantage of the creation of new models, corrects errors from previous old models, and uses gradient descent algorithms to minimize the loss of adding new models and ultimately combines all results for final prediction. Gradient Boosting supports model problems for regression prediction and classification prediction. The objective function of XGBoost consists of two parts. The first part is used to calculate the difference between the predicted score and the true score: Obj(t)= $_(i = 1)^n$ $L(y_i, ^(t-1)+f_t\ (x_i\ ))$ $+\Omega(f_t\ )$+constant The second part is normalization $\Omega(f_t\ )$, and the formula is as follows:

$\Omega(f_t\ )$= T+1/2 $_(i = 1)^T$ w_j^2

T represents the number of leaf nodes, w_j represents the weight of the j leaf nodes ， control the number of leaf nodes, control the score of the leaf nodes not too large to prevent over-fitting.

XGBoost uses the above objective function value as an evaluation function and applies the same concept as the CART regression tree. It uses a greedy algorithm to find all feature points and find the best segmentation point and best features. In addition, the maximum depth of the decision tree can be set at the same time, and the splitting is stopped to prevent overfitting when the sum of sample weights is less than the threshold. XGBoost fully supports the scikit-learn suite in Python

and supports random gradient enhancements. Subsampling can be done in each split level column, and all CPU cores can be built in parallel during model training so that XGBoost can demonstrate its powerful learning performance in a series of difficult Machine Learning tasks. In addition to XGBoost, which runs on a single host, it also supports distributed frameworks such as Hadoop and Spark. Therefore, organize a group of machines to train very large models. In this work, XGBoost will be used to implement Machine Learning and will predict the attack behavior of network log data.

2.6 Deep Learning

Deep Learning is not a completely new concept, but it has received renewed attention in recent years due to the advancement of computer hardware. IBM Watson, who won the puzzle quiz award in 2011,and Google AlphaGo, which defeat the World Go Chess in 2016 are the best endorsements. The so-called Deep Learning is a kind of "neural network" that imitates the neural network of the human body, and deepens its network. Deep Learning is a branch of Machine Learning. The Deep Learning has the ability to automatically extract features. Through the linear or non-linear transformation in the processing layer, the features in the data that are sufficient to represent the data will be extracted. Deep Learning involves many very complex statistical principles, it divided into four types: supervised learning, semi-supervised learning, unsupervised learning, and reinforcement learning. Each of the four learning styles has advantages and disadvantages, supervised learning will be used in this work, and this work will understand and use it as a tool to detect cyber attacks.

Next, introduce DNN. DNN means Deep Neural Networks. It is a popular topic in the field of machine learning in the industry and academia in recent years. The success of the DNN algorithm has raised the previous machine learning recognition rate to a new level. The generalized DNN contains variants such as CNN and RNN, and in practical applications, the so-called Deep Neural Networks

usually incorporates several known structures, such as LSTM or convolution layer. However, in a narrow sense, the difference between DNN and RNN and CNN is that DNN is especially expressed as a fully connected neuron structure, and does not contain convolution units or temporal associations. DNN is sometimes called Multi-Layer perceptron (MLP) However, DNN also has its limitations:

1. Huge number of parameters

   Since DNN uses a fully-connected neuron structure, the expansion of the number of parameters is often brought about in this structure, which not only easily leads to over-fitting, but also tends to cause local optimum.

2. Local optimum

   With the deepening of the neural network, the optimization function makes it easier for the model to fall into a locally optimal situation, deviating from the true global optimal solution. For situations where the amount of training data is limited, performance is even lower than for shallow networks.

3. Gradient disappearance or gradient explosion

   The gradient decays when using the sigmoid function in the backpropagation gradient. As the number of neural network layers increases, the gradient is infinitely close to zero at the bottom.

Next is RNN. The neural network used to process sequence data is called RNN. In the neural network model of DNN, the neural layers are fully connected, but the nodes of each layer are not connected. This neural network model is very inefficient in processing sequence problems. For example, in the prediction of advertising promotion, you need to understand the user's browsing habits or preferences and use it as a basis. The principle of the RNN model is to connect the output of the neuron back to the input of the neuron. The network memorizes the previous message and uses it for the calculation of the current output. That is to say, the output of the RNN is not only affected by the input of the previous layer, but also by the output of the same layer of neurons. RNN also has the following different applications.

1. One-to-One

   Fixed input and output lengths, such as the relationship of functions, it is generally not called a sequence problem, it is a mapping relationship.

2. One-to-Many

   Single input, multiple outputs. The textual description of the image is a One-to-Many problem. Enter a picture to output a sequence of text describing the image.

3. Many-to-One

   Multiple inputs, single output. Many-to-One applications, such as entering a paragraph of text, emotionally classify it.

4. Many-to-Many

   Multiple inputs, multiple outputs. A typical Many-to-Many application is language translation.

2.7 Related Works

Before starting this research, there are many of the theories, ideas, and experimental structures of our predecessors, which allowed us to have better results in our experiments. According to the background of this work [18], Iman Sharafaldin et al [27] gave us a lot of inspiration, also analyzed a large amount of data and visualized it, and proposed a classification of cyber attacks. In addition, at the IEEE International Conference on Smart Computing (SMARTCOMP) in 2017, a conference paper published by X. Yuan et al, mentioned [29] the defense mechanism of DDos and its use of Deep Learning to establish a DDoS attack, also given us inspired. In addition to these, there are many papers that give us a lot of constructive references [28].

In a paper published by Rafał Kozik et al [26], the flexibility of cloud-based architecture was used for large-scale Machine Learning, shifting high computing

requirements and high-storage parts to the cloud. The cloud first builds a complex learning model and then uses edge computing to execute it.

In a paper published by Muhammad Al-Qurishi et al [19], a model for predicting Sybil attacks using Deep Learning is proposed. The Sybil attack effectively denies the reception or transmission of real nodes on the network by creating enough false identities, effectively blocking the network services of other users. Through its experiments, it is possible to effectively provide high-precision predictions even when importing uncleaned data. The network log data used in this work is provided by the campus network security system, and the training and prediction are raw data. Through experiments, even complete attack behavior data can be highly accuracy without error judgment.

James Zhang et al, have proposed a method to detect abnormal behavior of network performance data [16], which uses Open Science Grid to collect and use perfSONAR servers and uses Boosted Decision Tree (BDT) and simple feedforward neural networks for Machine Learning. In this work, eXtreme Gradient Boosting is also used for decision classification to detect anomalous behavior in network log data. The network log data is divided into attack and non-attack, and finally submitted to ELK for visualization analysis.

As stated in the motivation in Chapter 1, today's hackers can use pollution training data to achieve classification that undermines Machine Learning and input design data into training data to reduce detection accuracy. The paper published by Sen Chen et al, proposes a two-stage learning enhancement method KUAFUDET [17] to learn and identify malware through confrontation detection. This includes the training phase of selecting and extracting features, and the testing phase of using the first phase of training. In the study of this work, the sorting extraction of feature importance was also used, and the complete attack data and the general original log data were imported as experimental data for reference comparison.

Hongyu Liu et al, have proposed a point-to-point detection method [20]. Based on the Deep Learning model of convolutional neural networks and recurrent neural networks, payload classification (PL-RNN) is performed and used for attack detection. XGBoost is used in this work to learn log data and summarize its important features. It effectively detects the difference between normal data and aggressive behavior and serves as the basis for both classifications.In addition, a paper published by Peiyuan Sun et al, [30] a Machine Learning-based approach was proposed, which can model the attack behavior based on intuitive observation.

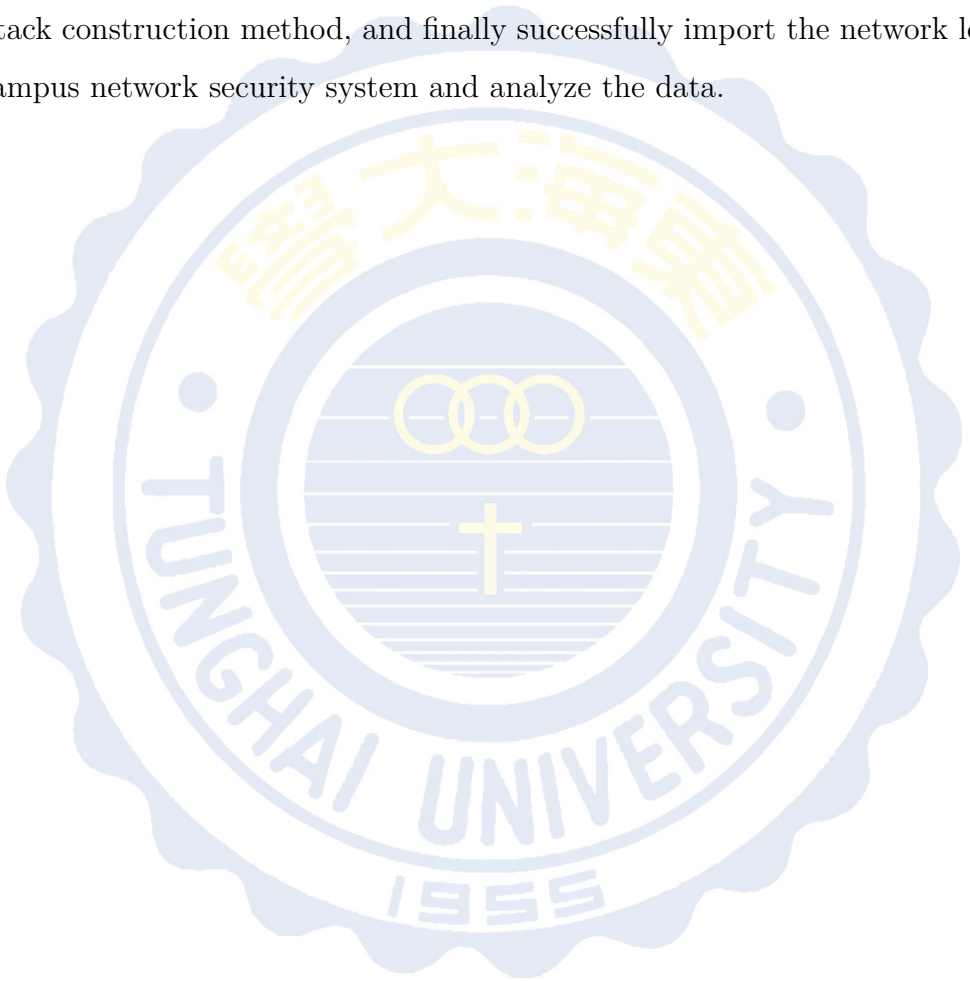
Ibrahim Ghafir et al, have proposed a Machine Learning-based system [22] that can detect and predict APT attacks accurately and quickly. The system can be evaluated experimentally and APT can be predicted in an early step. The prediction accuracy rate is 84.8%. In this work, Machine Learning is also used to quickly build a predictive model to classify network logs. It has half of the cyber attack behavior and has high accuracy. In addition, in this work has constructed a visualization system that provides network log data so that administrators can easily view log data at any point in time.

The paper presented by Ozgur Koray Sahingoz et al [25], mentions that phishing is one of the methods used by hackers today, and it proposes a real-time anti-phishing system, which has been experimentally proven to detect the network. Authentic rate of 97.98%when phishing URL

The paper presented by Abebe Abeshu Diro and Naveen Chilamkurti [23], mentions that applying Deep Learning for attack detection is the preferred approach because of its high feature extraction capabilities.In this work, also hope Machine Learning can make progress in detecting attacks.

At the 2015 International Conference on Information and Communication Technology and Systems (ICTS), P. P. I. Langi et al, presented an assessment of Logstah and Elasticsearch [21].The managers of the Institute of Nuclear Physica, Italy (INFN), used ELK Stack to set up a monitoring system to facilitate the management of each node's activities [24].In a conference paper, T. Ram Prakash

et al, proposed the construction of the ELK Stack system and how to geographically identify network users [15]. In addition, the paper by Chao-Tung Yang et al [14], also proposed a visual platform system using ELK Stack as a statistical analysis of air quality and influenza-like illness. This work refer to the ELK Stack construction method, and finally successfully import the network log of the campus network security system and analyze the data.



Chapter 3

System Design and Implementation

This chapter describes how to use artificial intelligence to build predictive models and use ELK Stack to visualize the implementation of system architecture and network log data. In addition, this work will build a Deep Learning model using DNN and RNN to compare with the XGBoost Machine Learning model. The network logs collected in this work are based on campus network devices, with more than 7 million data per day, approximately 2 to 3G. 2 TB has been collected and continues to increase.

3.1 System Architecture

In this work will install Anaconda3 on Window10 and use Juypter Notebook as the Python development environment. After pre-processing the network log data in the development environment, using XGBoost for Machine Learning and execute historical network log data to check the cyber-attack behavior. In addition, construct a network log system on Linux systems using open source software such as ELK Stack to visualize the cyber attack behavior for more intuitive analysis by managers.

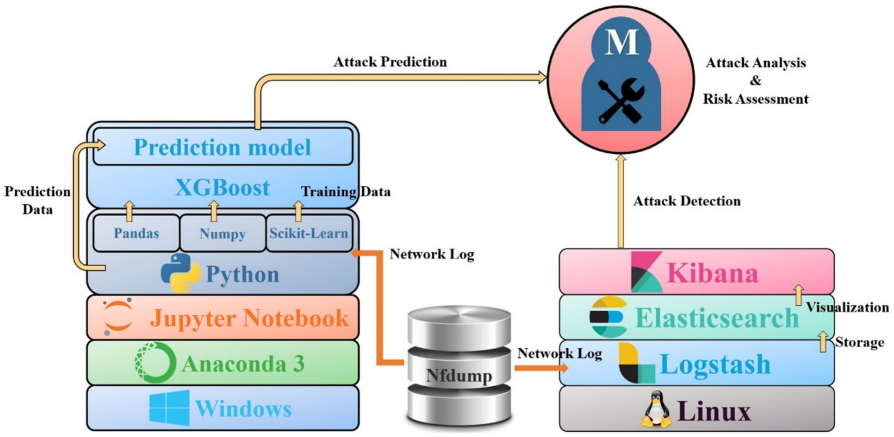


FIGURE 3.1: System architecture

As shown in Figure 3.1, the network logs are collected and submitted to the ELK for visual analysis to present the results of the cyber attack behavior detection to the administrator. On the other hand, Python, which imports network logs, performs data preprocessing and performs model training. Finally, the model submits the cyber attack prediction result to managers. If the ELK Stack analyzes the log data into a normal data stream, but the model prediction results show that the data stream is an attack behavior, the administrator can use the results of both parties for cross-validation analysis to perform risk assessment. It can prevent the impact of hidden cyber attacks or unknown cyber attacks.

3.2 NetFlow Log System

First, using Linux built-in shell scripts to write scripts and schedules, so that the machine can automatically download the network log data from the server side. After data processing, Logstash collects and filters the log data, and then Logstash is transferred to Elasticsearch for later data search or analysis, and then Kibana is used to visualize the analyzed data and finally present it on the website. The above is the NetFlow Log System, a campus network log platform.

3.2.1 Network Usage

Before analyzing the cyber attack behavior, this work can set up several frequently used domains and visualize the log data, from which the administrator can monitor the network for abnormal use. In addition, in this paper, these domains are divided into search engines, auction sites, online communities, entertainment, and high-risk domains. All of the above domain IPs are public IP and can only be observed by the administrator.

3.2.2 Attack Data Analysis

Cyber attacks tend to hide their packaging and pretend to be a secure data stream to trick the information security system. However, just like walking in the snow, we will definitely leave footprints. In this work selected several kinds of cyber attacks and recorded their eigenvalues. Then use Elasticsearch to filter the cyber log data. Data visualization of suspected cyber attacks is monitored by managers.

3.3 Machine Learning with XGBoost

In this section, discuss how to use XGBoost for Machine Learning and construc a dichotomy prediction model to detect the cyber attack behavior in network log data and determine which data streams in the network log data are suspected of having cyber attacks behavior, and which are normal. XGBoost has the following advantages, which are why chose it:

1. Provides many options to prevent overfitting, such as: normalization, maximum depth of the tree, and subsampling by column.

2. XGBoost supports parallel construction. After selecting the best classification point, it can calculate the node gain of the same level in parallel, and the training speed is fast. Have the processing of sparse data.

3. Cross-validation, early stop, when the number of training reaches the pre-set number of iterations and the prediction result is not in the ascension, the training is stopped early to avoid over-fitting and speed up the training.

4. Support setting sample weights, even if the data samples are not balanced, can train the model by adjusting the weights.

3.3.1 Data Preprocessing

First, the log data must be preprocessed to convert the data to a format that the machine can learn. The algorithm is as follows.1 In addition, our log data has an average of about 500,000 data per data, and the data of suspected aggression accounts for about 1.8% of the total number of single log data.

Algorithm 1 Data Preprocess for Prediction model

Input: Nfdump.txt log data from the campus network environment, *Dataset*;
Output: *Training*, *Test*, and *Prediction* Dataset;

Python read *Dataset*;
Let column in *Dataset* = *data feature*;
Del *unneeded data* and *blanks*;
if (data = cyber attacks behavior) **then**
    mark data = *Attack* = 1;
else(data != cyber attacks behavior)
    mark data = *Normal* = 0;
end if
return Dataset;
Training Dataset = *Split* 66%*Dataset*;
Test Dataset = *Split* 33%*Dataset*;
Prediction Dataset = *full Dataset*;

3.3.2 XGBoost Model Training

Undertake the preprocessing data of 3.3.1, and then import the data into the model for Machine Learning training. However, compared to data with cyber-attacks, normal network usage data accounts for the vast majority of the logs, and may not even appear at all. Therefore, how to make the model learn the correct features is the primary goal.

In order to avoid Machine Learning to classify data, the data of cyber-attack behaviour is classified as normal traffic or noise. Therefore, collecting log data for multiple time periods and filter out the data with attack characteristics to form a training set. Our training set will try to train by writing data from different attacks and non-attacks. Finally, in this work, both attack and non-attack data contain approximately 50% of the data, providing the best model feedback. The training set contains a total of approximately 150,000 log data, and the validation set contains 50% of the data, including attack data and non-attack data, for a total of approximately 77,000 data. In addition, useing random floating parameters to adjust the parameters in XGBoost, use L1 and L2 normalization to perform regular gradient enhancement, avoid overfitting or inappropriate, and the feature importance is passed after each training to adjust the characteristics of the log data.

3.3.3 XGBoost Model Prediction

In the forecast set, use two types of data to import Machine Learning model predictions. The first is data consisting of 100% complete attack data, and the second is new unmodified log data. This verifies the correctness and versatility of our model. Finally, the training and validation of the model is completed, which will have high precision and good F1 score. The algorithm is as follows.2

Algorithm 2 The Prediction model for cyber attacks

Input: *New raw log Data* from $Nfdump.txt$;
Output: The *amount* of predicted data for cyber attacks;
1: Upload $Nfdump.txt$ to website;
2: Python read *New raw log Data*;
3: *New raw log Data* do Data Preprocess;
4: Load *Prediction model*;
5: **if** (*data* = cyber attacks behavior) **then**
6: *Count data* = *Attack*;
7: **else**(*data* != cyber attacks behavior)
8: *Count data* = *Normal*;
9: **end if**
10: *return Count*;
11: *validation accuracy*

3.4 Deep Learning with Keras

In this chapter, discuss how to use Keras for Deep Learning, In the field of Deep Learning, CNTK and TensorFlow are widely used in Deep Learning research. Although both have very powerful features, the actual application is more difficult. Therefore, the Deep Learning project for this job will use Keras to build a dichotomy prediction model, and perform the cyber attack behavior detection on network log data, and determine which data streams in the network log data are suspected of having the cyber attack behavior, and which are normal. In this work, the DNN model and the RNN model will be built, and they will be experimented with the same data as XGBoost Machine Learning model.

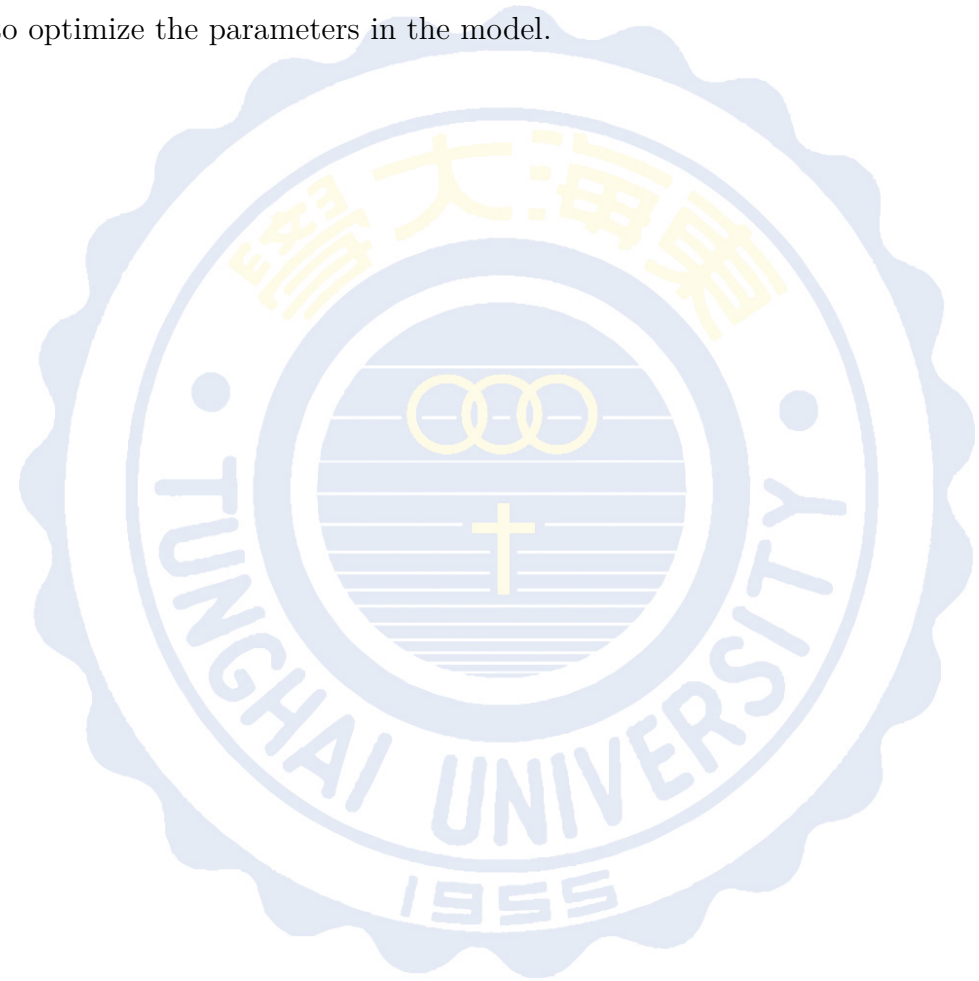
3.4.1 Deep Neural Networks Model

First, the log data is pre-processed as in Section 3.3.1, and the data is converted into a format that the machine can learn. After the data is imported into the DNN model, it is trained in a supervised learning manner. In order to ensure that the DNN model can produce a global optimal solution during the experiment, this work uses the scikit-learn suite to optimize the parameters in the model. Since the purpose of this work is to predict potential attacks, there are several types of attacks in the data set. However, the characteristics of cyberattack behavior are very scattered, which leads to the problem of over-fitting or gradient disappearing even after numerous adjustments. In order to ensure the fairness of the experiment, the data set is also given a full attack data set, as well as a new unmodified log data resource for validation.

3.4.2 Recurrent Neural Network Model

In addition to the DNN solution, the data problem in the Deep Learning mode also has RNN. Since DNN cannot fully predict full attack data, and often there is over-fitting or gradient disappearance, RNN can also deal with data problems

and can greatly improve the problem of DNN over-fitting. Therefore, this work is connected to the DNN model to rebuild an RNN model and give the same data to conduct experiments. In order to ensure that the RNN model can produce a global optimal solution during the experiment, this work uses the scikit-learn suite to optimize the parameters in the model.



Chapter 4

Experimental Results

This section describe the use of XGBoost to build a Machine Learning model, and using Keras to build DNN and RNN Deep Learning models for binary classification prediction, and how to use ELK Stack to analyze network usage and attack behavior characteristics. In Section 4.1, introduced the implementation of the experimental environment and the NetFlow Log. Sections 4.2 through 4.3 show the results of the ELK Stack analysis network log visualization. Section 4.4 to 4.5 are the data preprocessing and prediction results of the XGBoost model. At last, Section 4.6 to 4.7 are the prediction results of the Deep Learning model DNN and RNN.

4.1 Experimental Environment

This section describes our hardware lab environment. This experiment uses two hosts, one with Linux as the operating system, as the server that sets up the ELK Stack. The other is to use Window10 as the operating system, install Anaconda 3 and related kits as the Python development environment, and build the XGBoost Machine Learning model. Detailed hardware devices are shown in Table 4.1.

TABLE 4.1: Hardware specifications

Item	Disk	Core	Ram	OS
NetFlow Log	8TB	10 CPUs x Intel(R)Core(TM) i7-6950X CPU @ 3.00GHz	128G	Ubuntu 18.04
Machine Learning	1TB	Intel(R)Core(TM) i7-7700 CPU @ 3.60GHz	16G	Windows 10

4.2 ELK Stack Network Usage

In order to more easily confirm the network usage on campus, this experiment finds the public IP addresses of major commercial websites, search engines, social networks, etc. through the Internet. This information can be easily found on websites such as ipinfo.io. Use Elasticsearch to filter the required domain information, remove the non-service local IP address to avoid information miscellaneous, record the required domain name, and use Kibana to visualize it. Figure 4.1 shows the pie chart of Network usage.

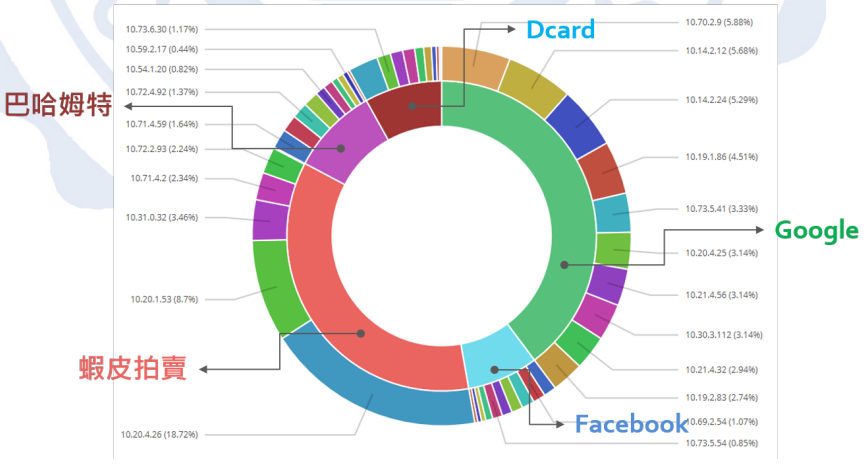


FIGURE 4.1: Network usage

4.3 ELK Stack Attack Analysis

In this experiment, the characteristics of several kinds of the cyber attack behaviors are selected as the screening conditions, and after ELK analysis, the data is visualized and presented, which provides an intuitive way for the administrator

to observe the cyber attack behavior. As Figure 4.2, Figure 4.3, Figure 4.4, and Figure 4.5 shown below.

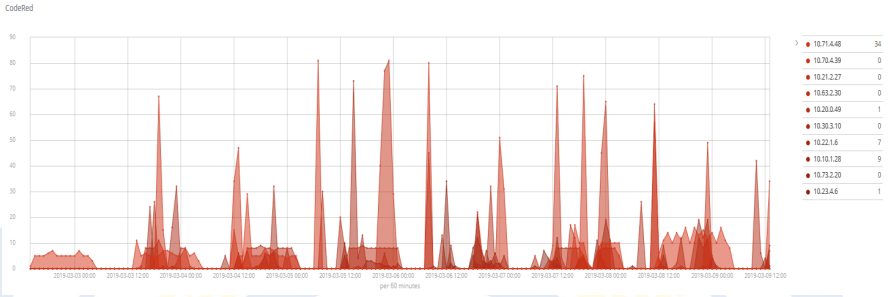


FIGURE 4.2: CodeRed

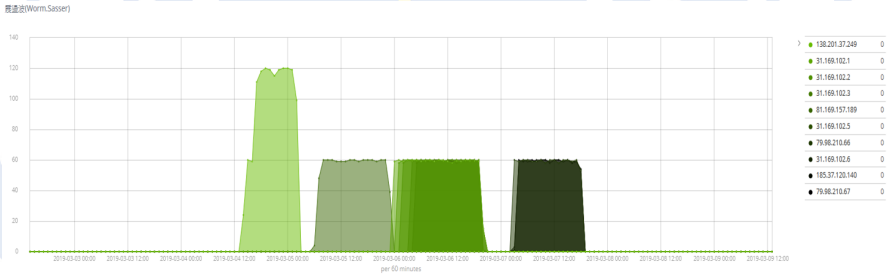


FIGURE 4.3: Worm Sasser

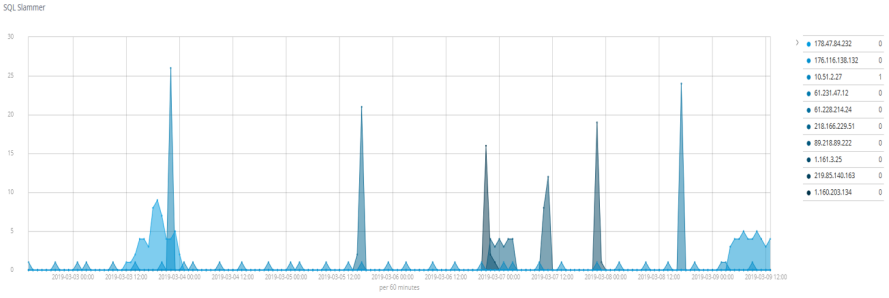


FIGURE 4.4: SQL Slammer

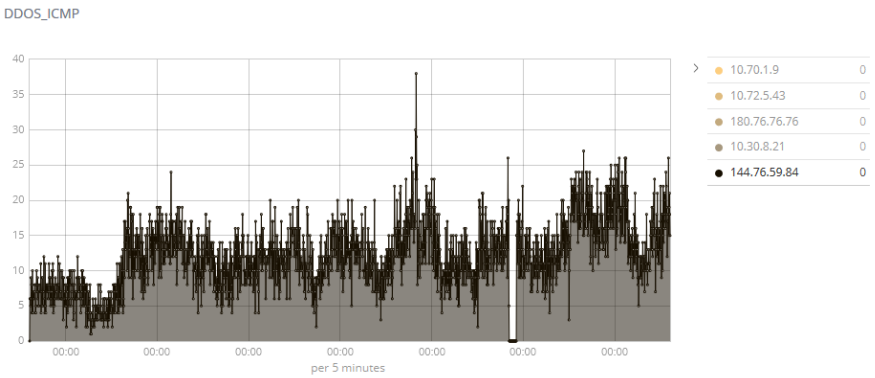


FIGURE 4.5: ICMP DDOS

4.4 Machine Learning Data Preprocessing

In the network log data of the campus environment, there are about 500,000 data per data, and the data of suspected aggression accounts for about 1.8% of the total. In order to achieve better training conditions, the experiment will have the best training results after about 50% of each experimental attack and non-attack. Therefore, this experiment uses ELK Stack to filter the attack data of other time periods, and then extract the log data from the database for integration. Finally, the log data is pre-processed to complete the pre-operation of the training set and the verification set. A total of about 200,000 pieces of data will be divided into 66% as a training set and 33% as a verification set. Figure 4.6 shows the data description.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227992 entries, 0 to 227991
Data columns (total 9 columns):
Dst Pt      227992 non-null int64
Duration    227992 non-null float64
In Byte     227992 non-null int64
In Pkt      227992 non-null int64
Input       227992 non-null int64
Output      227992 non-null int64
Proto       227992 non-null int64
Src Pt      227992 non-null int64
attack      227992 non-null int64
dtypes: float64(1), int64(8)
memory usage: 15.7 MB
```

FIGURE 4.6: Data description

4.5 XGBoost Model Prediction

As shown in Figure 4.7, the XGBoost parameter is adjusted several times and optimized by using floating parameters to obtain the parameter values.

```
model = xgb.XGBClassifier(learning_rate = 0.008597735392699073, objective = 'binary:logistic',
                          n_estimators = 472, n_jobs = 4,
                          max_depth = 4, min_child_weight = 8,
                          seed = 30, max_delta_step = 6,
                          subsample = 0.7065226970496071, colsample_bytree = 0.515980616951696,
                          gamma = 0.044644589297517134, scale_pos_weight = 1,#46
                          reg_alpha = 80, reg_lambda = 38)
```

FIGURE 4.7: XGBoost adjusted model

As shown in Figure 4.8, the training set is imported into the XGBoost model for training, and the initial accuracy rate is 98.24%.

```
eval_set = [(X_train, y_train),(X_test, y_test)]
model.fit(X_train, y_train,
          early_stopping_rounds=49,
          eval_metric=["rmse","auc","error", "logloss"],
          eval_set=eval_set,
          verbose=True)
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
[466]   validation_0-rmse:0.129213      validation_0-auc:0.994532       validation_0-error:0.017859     validation_0-logloss:
0.082595        validation_1-rmse:0.129042      validation_1-auc:0.994719       validation_1-error:0.017597     validation_1-
logloss:0.082268
[467]   validation_0-rmse:0.129115      validation_0-auc:0.994561       validation_0-error:0.017859     validation_0-logloss:
0.082408        validation_1-rmse:0.128945      validation_1-auc:0.994744       validation_1-error:0.017597     validation_1-
logloss:0.082083
[468]   validation_0-rmse:0.129034      validation_0-auc:0.994562       validation_0-error:0.017846     validation_0-logloss:
0.082227        validation_1-rmse:0.128861      validation_1-auc:0.994745       validation_1-error:0.017597     validation_1-
logloss:0.081902
[469]   validation_0-rmse:0.128939      validation_0-auc:0.994568       validation_0-error:0.017846     validation_0-logloss:
0.082041        validation_1-rmse:0.128766      validation_1-auc:0.99475        validation_1-error:0.017597     validation_1-
logloss:0.081711
[470]   validation_0-rmse:0.128907      validation_0-auc:0.994571       validation_0-error:0.017846     validation_0-logloss:
0.082002        validation_1-rmse:0.128736      validation_1-auc:0.994754       validation_1-error:0.017597     validation_1-
logloss:0.081667
[471]   validation_0-rmse:0.128884      validation_0-auc:0.994594       validation_0-error:0.017846     validation_0-logloss:
0.081953        validation_1-rmse:0.128711      validation_1-auc:0.994775       validation_1-error:0.017597     validation_1-
logloss:0.081627
Accuracy: 98.24%
```

FIGURE 4.8: Data training

XGBoost can derive the feature gain score from the model training results and calculate the feature importance score to the developer, as shown in Figure 4.9.

```
importance_plot = pd.DataFrame({'feature':list(feature_names),'importance':model.feature_importances_})
importance_plot = importance_plot.sort_values(by='importance')
importance_plot = importance_plot.reset_index(drop=True )
thresholds = importance_plot.importance
thresholds_valid = np.unique(thresholds[thresholds != 0])

thresh = 0.028
selected_features = list(importance_plot[importance_plot.importance > thresh]['feature'])
print('selected features are :\n %s'%selected_features)
```

```
selected features are :
 ['Proto', 'Dst Pt', 'Input', 'Output']
```

```
print('features score :\n ', model.feature_importances_)
```

```
features score :
  [0.00433323 0.01490925 0.19005358 0.12568218 0.02094196 0.02757531
 0.22661725 0.38988718]
```

```
Thresh=0.004, n=8, Accuracy: 98.57%
Thresh=0.015, n=7, Accuracy: 98.57%
Thresh=0.021, n=6, Accuracy: 98.25%
Thresh=0.028, n=5, Accuracy: 98.21%
Thresh=0.126, n=4, Accuracy: 98.02%
Thresh=0.190, n=3, Accuracy: 98.01%
Thresh=0.227, n=2, Accuracy: 89.91%
Thresh=0.390, n=1, Accuracy: 89.91%
```

FIGURE 4.9: Feature score

Fig. 4.10 is a bar graph in which the feature importance is sorted according to the score. In order "Dst Pt","In Byte","Src Pt","Output","In Pkt ","Duration","Proto","Input".

```
ax = xgb.plot_importance(model, height=0.5)
fig = ax.figure
fig.set_size_inches(12,7) #可調節圖片尺寸和緊密程度
pyplot.show()
#Duration[0] Src Pt[1] Dst Pt[2] Proto[3] In Pkt[4] In Byte[5] Input[6] Output[7]
```

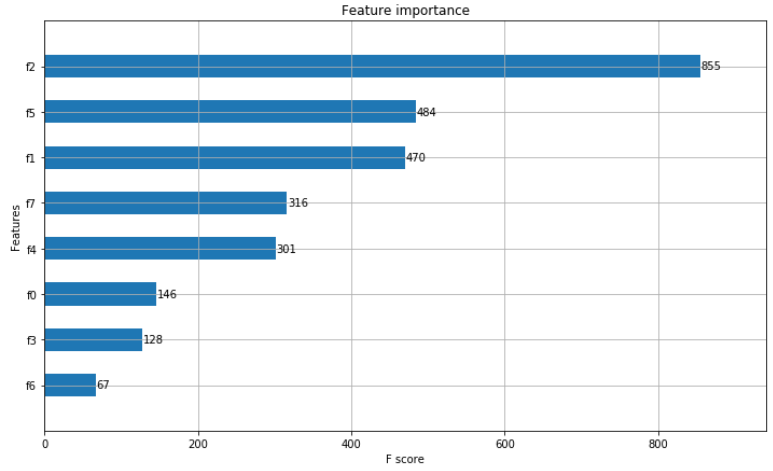


FIGURE 4.10: Graph of feature score

As shown in Figure 4.11, Gain represents the relative contribution of the feature to the model, and a high value means that it is more important for prediction.

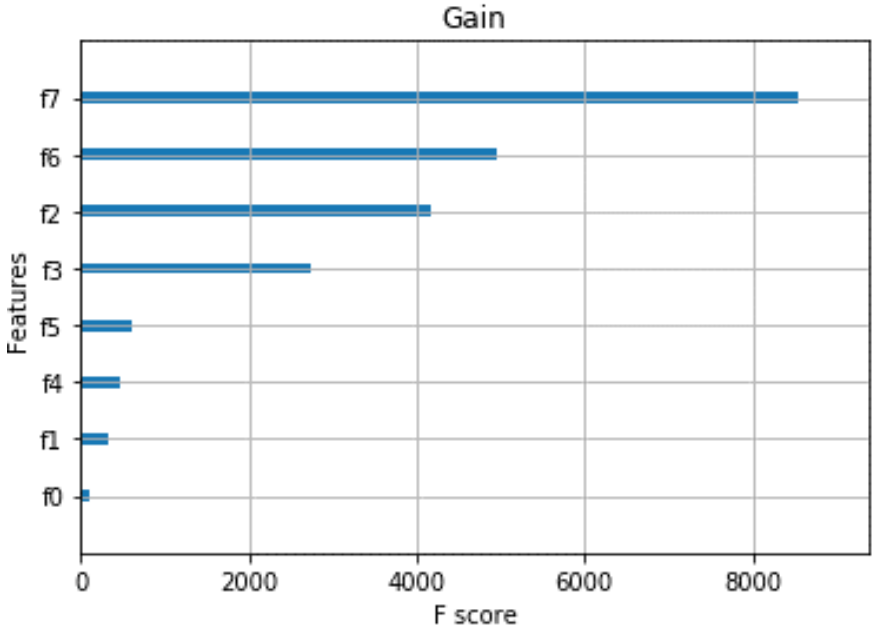


FIGURE 4.11: Gain pyplot

Weight indicates the number of times the feature is used to split the node. As shown in Figure 4.12.

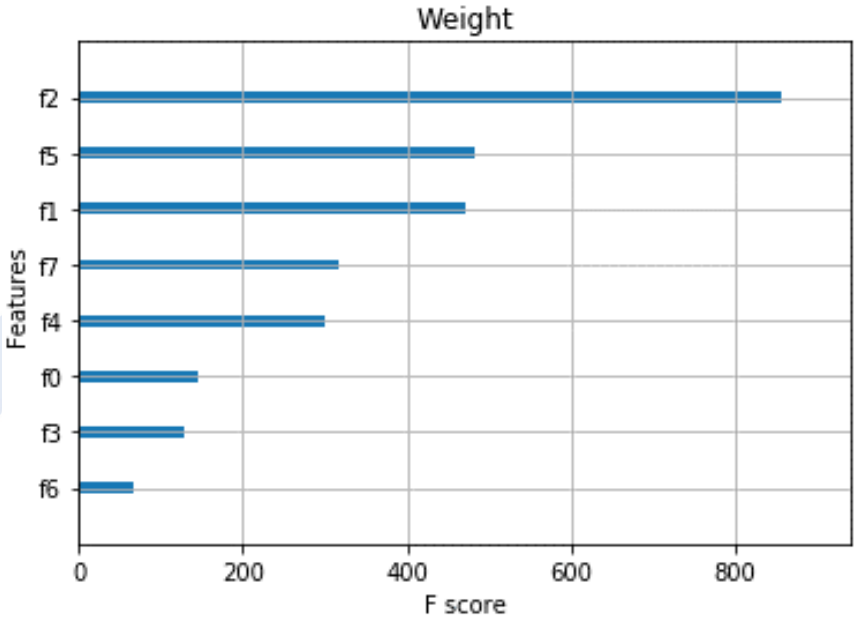


FIGURE 4.12: Weight pyplot

Cover in Fig. 4.13 represents the relative number of observations associated with this feature, for example, 100 observations, 4 features, and 3 trees, assuming f1 is used to determine 10, 5, and 2 observations in t1, t2, and t3, respectively. Calculate the coverage of this feature as $10 + 5 + 2 = 17$ observations.

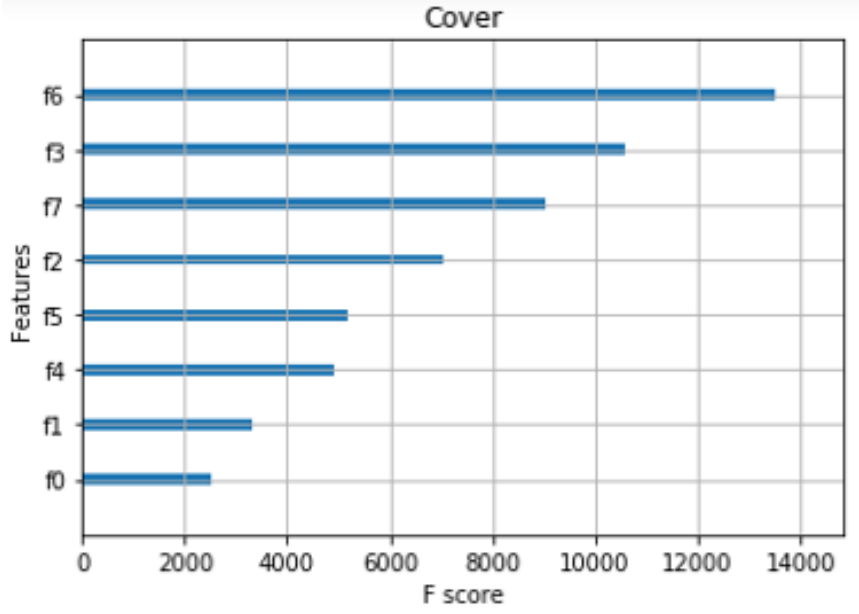


FIGURE 4.13: Cover pyplot

Total Gain represents the total gain that a feature brings in each split node in all trees as shown in Fig. 4.14. The number of all samples covered by a feature at each split node is called Total Cover as shown in Figure 4.15.

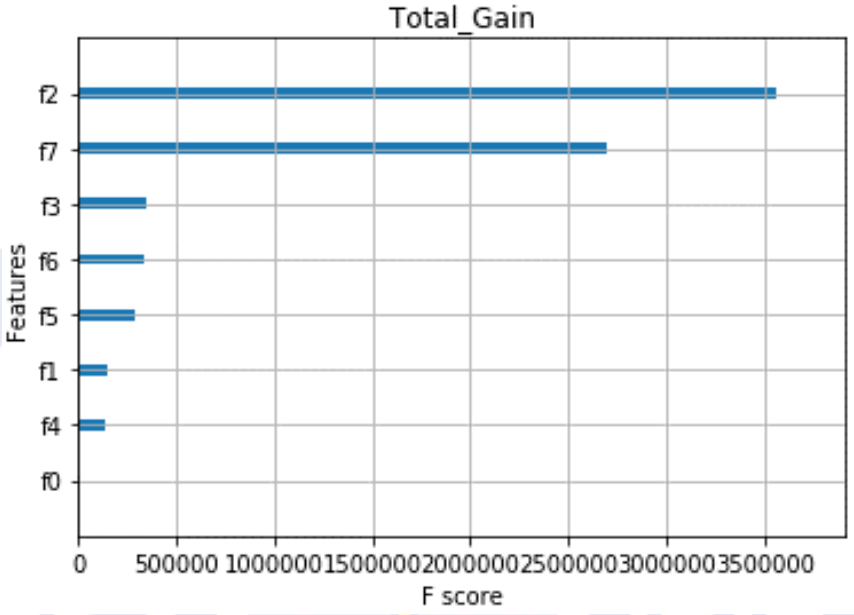


FIGURE 4.14: Total Gain pyplot

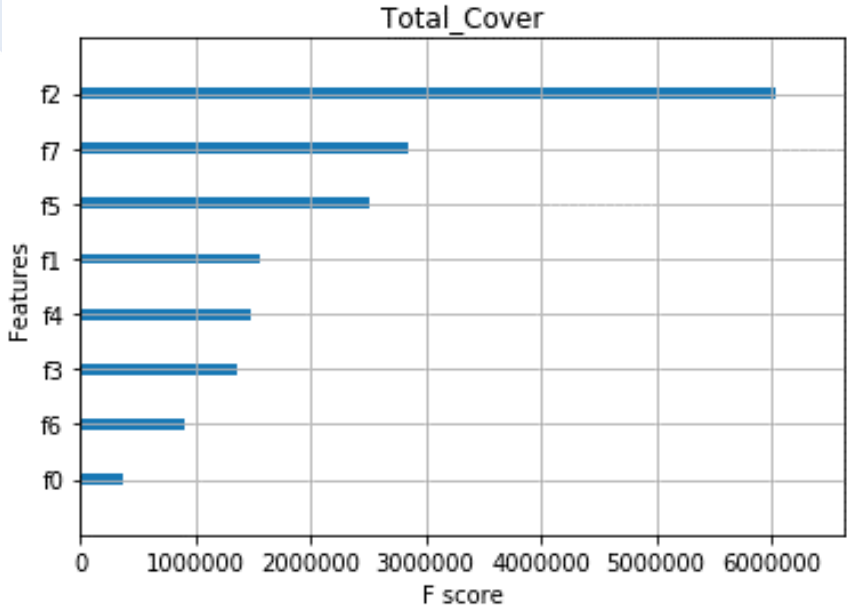


FIGURE 4.15: Total Cover pyplot

Figure 4.16 shows the decision tree of XGBoost. Can clearly understand the logic that the decision tree wants to express. Leaf refers to the predicted value, and those probability values associated with the leaf nodes represent the conditional probability that a particular branch of a given tree reaches the leaf node. The branches of the tree can be presented as a set of rules.

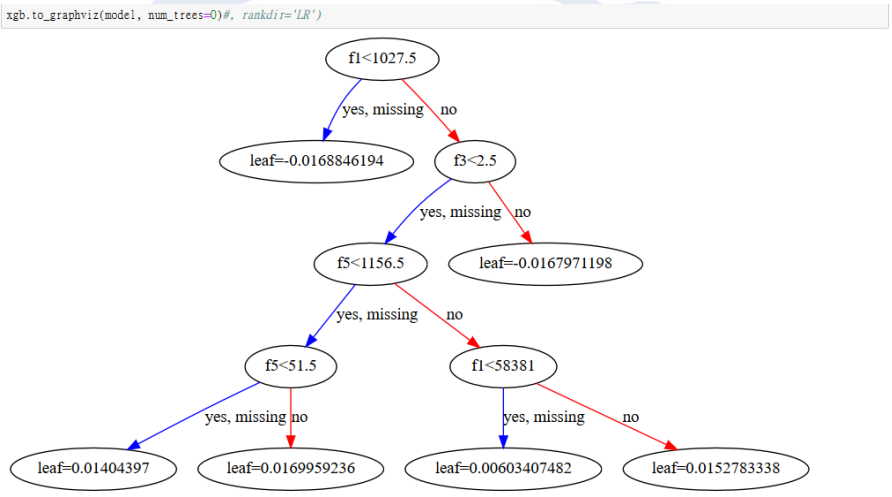


FIGURE 4.16: XGBoost decision tree

In the end, the prediction, as described in Chapter 3, in order to verify the correctness and versatility of the model, the data used in the prediction is the new raw log data, and the clean data is handed over to the model after preprocessing. The predicted result is as high as 96.01%. In order to verify the correctness of the model, a set of full-attack prediction sets is resampled here, and the accuracy rate is as high as 100%. This proves that the attack data can be fully recognized when there are attack behavior characteristics in the log data. As shown in Tabel4.2.

TABLE 4.2: Model predictions

Predictions Dataset	**Predictions Accuracy**	**Predictions Count**		
New raw log data	**96.01%**	**Item**	**Attack**	**Normal**
		Val label	**6,342**	**127,424**
		Model Preds	**11,679**	**122,087**
Full attack log data	**100%**	**Item**	**Attack**	**Normal**
		Val label	**5,679**	**0**
		Model Preds	**5,679**	**0**

Finally, the Scikit-Learn built-in evaluation indicator suite is applied to test the model's mean square error (MSE), model accuracy, and F1 Score model correctness. As shown in Tabel 4.3. All have good results.

TABLE 4.3: Model score

Evaluation index	Score
MSE	**2.53%**
Accuracy	**97.47%**
F1 Score	**97.54%**

4.6 DNN Model Prediction

Figure 4.17 is the DNN model established by this work. Including an input layer and the final output layer, it also contains two hidden layers and three dropout layers, which are fully connected. All parameters seek the best solution by scikit-learn.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               1024
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
=================================================================
Total params: 67,073
Trainable params: 67,073
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4.17: DNN model

The DNN model is doing data training. Figure 4.18 shows the state of training and the time spent.

```
Train on 136795 samples, validate on 91197 samples
Epoch 1/50
 - 6s - loss: 3.4059 - acc: 0.5392 - val_loss: 0.8710 - val_acc: 0.7079
Epoch 2/50
 - 5s - loss: 2.2504 - acc: 0.6048 - val_loss: 0.5931 - val_acc: 0.8547
Epoch 3/50
 - 5s - loss: 1.7536 - acc: 0.6485 - val_loss: 0.5794 - val_acc: 0.8560
Epoch 4/50
 - 5s - loss: 1.4976 - acc: 0.6768 - val_loss: 0.5525 - val_acc: 0.8533
Epoch 5/50
 - 5s - loss: 1.3326 - acc: 0.6984 - val_loss: 0.5292 - val_acc: 0.8584
Epoch 6/50
 - 5s - loss: 1.2115 - acc: 0.7152 - val_loss: 0.5038 - val_acc: 0.8766
Epoch 7/50
 - 5s - loss: 1.1188 - acc: 0.7297 - val_loss: 0.4745 - val_acc: 0.8809
Epoch 8/50
 - 5s - loss: 1.0132 - acc: 0.7461 - val_loss: 0.4635 - val_acc: 0.8855
```

FIGURE 4.18: DNN model trainging

Figure 4.19 shows the training and validation loss values for this DNN model. It can be seen from the figure that the loss value of the training data keeps decreasing and is infinitely close to the validation data.

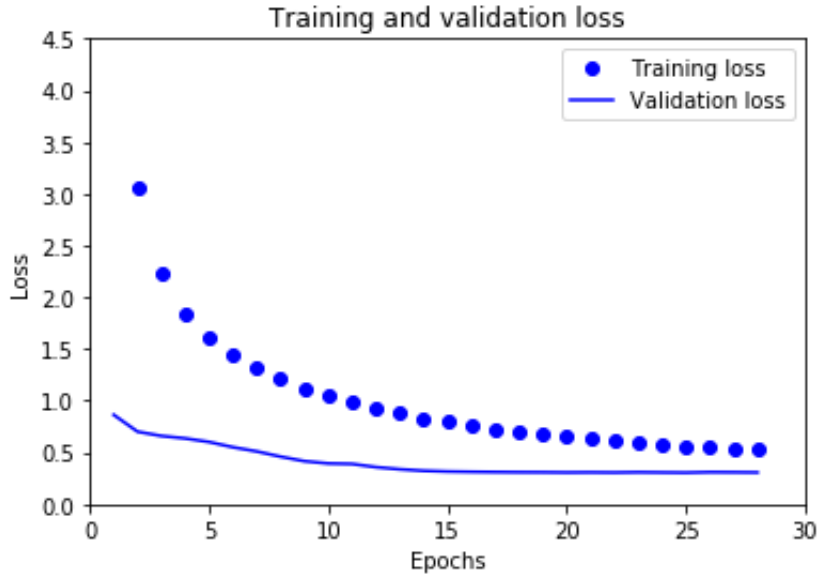


FIGURE 4.19: DNN training and validation loss

Figure 4.20 shows the training and validation accuracy values for this DNN model. From the figure that the accuracy of the training set is constantly increasing and close to the verification set. This is a good model.

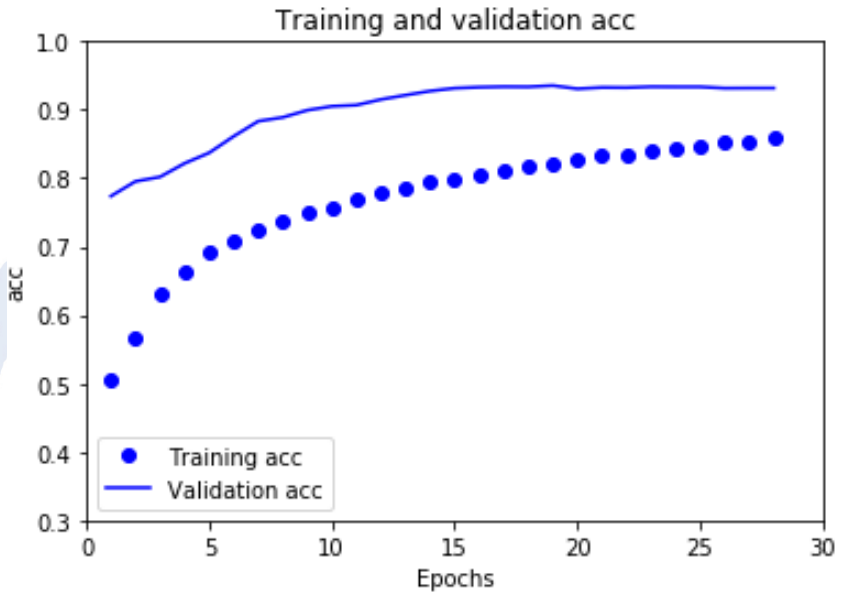


FIGURE 4.20: DNN training and validation accuracy

The model validation set prediction results are shown in Tabel 4.4. The data used for prediction is the same as the data used by XGBoost. The DNN model predicts results as high as 96.89%. In order to verify the versatility of the model, a set of full attack prediction sets is also sampled here, with an accuracy of only 69.66%.Compared with the previous accuracy record, it is very unexpected for such a result. The experimental result of this DNN model is the best.

TABLE 4.4: DNN model predictions

Predictions Dataset	**Predictions Accuracy**	**Predictions Count**		
Y_test data	**93.18%**	**Item**	**Attack**	**Normal**
		Val label	**45587**	**45610**
		Model Preds	**45711**	**45486**
New raw log data	**96.89%**	**Item**	**Attack**	**Normal**
		Val label	**567122**	**4373**
		Model Preds	**558100**	**13395**
Full attack log data	**69.66%**	**Item**	**Attack**	**Normal**
		Val label	**5679**	**0**
		Model Preds	**3956**	**1723**

4.7 RNN Model Prediction

Figure 4.21 is the RNN model established by this work. The difference between the RNN model and the DNN model is that the output of the RNN is not only affected by the input of the previous layer, but also by the output of the same layer of neurons. All parameters seek the best solution by scikit-learn.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, None, 128)         17408
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 128)         0
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, None, 256)         98560
_________________________________________________________________
dropout_2 (Dropout)          (None, None, 256)         0
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 128)               49280
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 165,377
Trainable params: 165,377
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4.21: RNN model

Same as DNN, figure 4.23 the time spent on model training, as well as the training and validation loss values and accuracy for this RNN model.

```
Train on 136795 samples, validate on 91197 samples
Epoch 1/50
 - 9s - loss: 3.8670 - acc: 0.5512 - val_loss: 0.8692 - val_acc: 0.8406
Epoch 2/50
 - 7s - loss: 2.2942 - acc: 0.6131 - val_loss: 0.4525 - val_acc: 0.8930
Epoch 3/50
 - 8s - loss: 1.6782 - acc: 0.6609 - val_loss: 0.4480 - val_acc: 0.8845
Epoch 4/50
 - 8s - loss: 1.4121 - acc: 0.6896 - val_loss: 0.4521 - val_acc: 0.8768
Epoch 5/50
 - 7s - loss: 1.2664 - acc: 0.7058 - val_loss: 0.4394 - val_acc: 0.8796
Epoch 6/50
 - 7s - loss: 1.1491 - acc: 0.7212 - val_loss: 0.4295 - val_acc: 0.8826
Epoch 7/50
 - 7s - loss: 1.0653 - acc: 0.7350 - val_loss: 0.4105 - val_acc: 0.8863
```

FIGURE 4.22: RNN model training

Figure 4.23 shows the training and validation loss values for this RNN model.Figure 4.24 shows the training and validation accuracy values for this RNN model. It can be seen from these two figures that this RNN model is also training in a good direction, and to a good accuracy rate.

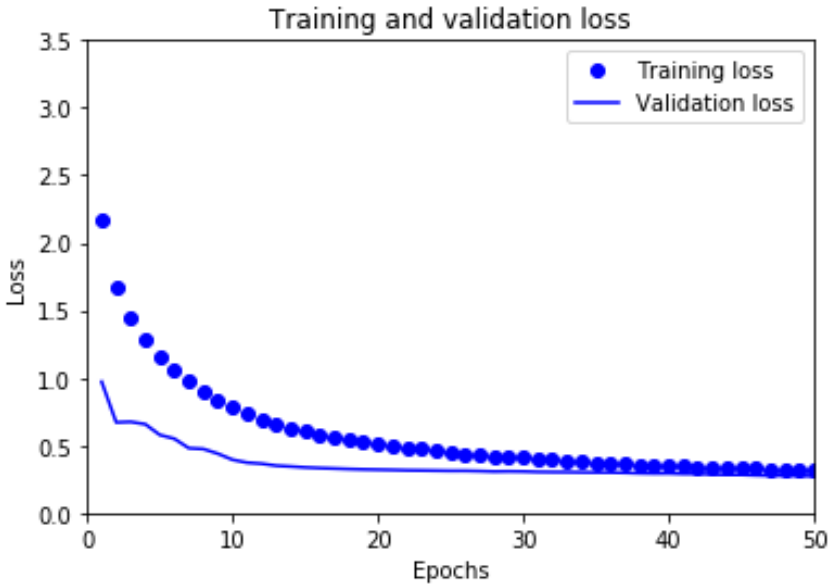


FIGURE 4.23: RNN training and validation loss

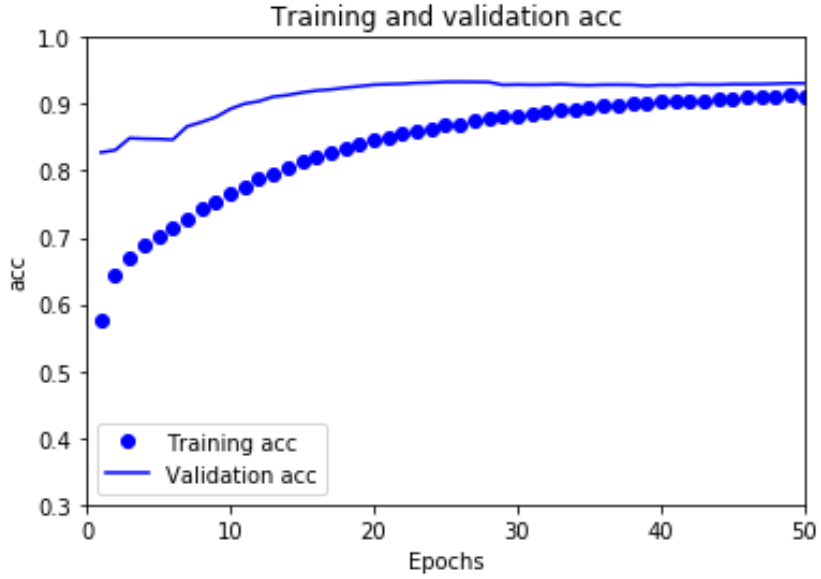


FIGURE 4.24: RNN training and validation accuracy

The prediction results of the RNN model are shown in Tabel 4.5. The data used for prediction is the same as the data used in the first two models. The RNN model predicts results as high as 97.61%, even surpassing the accuracy of XGBoost. In order to verify the versatility of the model, the same set of attack data were also used for prediction, but the accuracy was only 70.85%. However, the results of one of the best experiments compared to DNN are still slightly improved.

TABLE 4.5: RNN model predictions

Predictions Dataset	**Predictions Accuracy**	**Predictions Count**		
Y_test data	**93.34%**	**Item**	**Attack**	**Normal**
		Val label	**45587**	**45610**
		Model Preds	**45786**	**45411**
New raw log data	**97.61%**	**Item**	**Attack**	**Normal**
		Val label	**567122**	**4373**
		Model Preds	**562197**	**9298**
Full attack log data	**70.85%**	**Item**	**Attack**	**Normal**
		Val label	**5679**	**0**
		Model Preds	**4024**	**1655**

Chapter 5

Conclusions and Future Works

5.1 Conclusions

This paper introduces a network log system built using ELK Stack, which allows administrators to easily visualize charts and see the information they need from tens of millions of data. In terms of security protection, applied XGBoost construct Machine Learning model, and successfully identified the attack characteristics in the data, and also had a high accuracy rate of 96% in the prediction set. In this work also compare Machine Learning with Deep Learning models, shown in Tabel 5.1. From the experimental results, XGBoost won an overwhelming victory in the data prediction of the full attack. If the model is unable to fully detect the attack signature in the full attack data set, how do you ensure that the amount of abnormal data predicted in the general data is correct? Therefore, after the experimental results were presented, this work chose to use XGBoost as the Machine Learning model for the log data attack prediction. This attack prediction model can help us detect the ELK as the analyzed data. If the ELK Stack analyzes a log as normal data, and the model prediction results show that these data streams have aggressive behavior characteristics, the manager can use the two-party results to cross verification, further risk assessment of information security.

TABLE 5.1: Comparison of experimental results of three models

Predictions Accuracy			
Model	**XGBoost**	**DNN**	**RNN**
New raw log data	**96.01%**	**96.89%**	**97.61%**
Full attack log data	**100%**	**69.66%**	**70.85%**

5.2 Future Works

ELK Stack will collect more functional values related to the attack behavior and further visualize the Network log data as an analysis chart. Network usage will add the remaining large domain IP domains to it and distinguish each different domain. Convenient for management to observe. XGBoost is one of the most popular Machine Learning models, and I think its limitations are not limited to the two categories of attack and non-attack log data. In the future, hope to more actively increase the data characteristics of the attack behavior, enrich our database, and continue to study XGBoost, use XGBoost to create a multi-classification model, can directly identify the type of attack, and can find unusual data from the network log, and Distinguish between non-attack data. In addition, cross-validation can be used in conjunction with Deep Learning to compare predictions and improve information security.

References

[1] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3): 660–674, 1991.

[2] Eighty two percent of security professionals fear artificial intelligence attacks against their organization, 2018. `https://www.home.neustar/about-us/news-room/press-releases/2018/NISCOctober`.

[3] Neustar research finds global attacks like wannacry and goldeneye directly affect enterprise' cyber protection choices, 2017. `https://www.home.neustar/about-us/news-room/press-releases/2017/NISCSurvey`.

[4] Ashima Rattan, Navroop Kaur, and Shashi Bhushan. Standardization of intelligent information of specific attack trends. In *Progress in Advanced Computing and Intelligent Engineering*, pages 75–86. Springer, 2019.

[5] Marian Stewart Bartlett, Gwen Littlewort, Mark Frank, Claudia Lainscsek, Ian Fasel, and Javier Movellan. Recognizing facial expression: machine learning and application to spontaneous behavior. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 568–573. IEEE, 2005.

[6] Hayit Greenspan, Bram Van Ginneken, and Ronald M Summers. Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. *IEEE Transactions on Medical Imaging*, 35(5): 1153–1159, 2016.

[7] Marcin Bajer. Building an iot data hub with elasticsearch, logstash and kibana. In *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 63–68. IEEE, 2017.

[8] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[9] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[10] Sebastian Raschka and Vahid Mirjalili. *Python machine learning*. Packt Publishing Ltd, 2017.

[11] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[12] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[13] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[14] Chao-Tung Yang, Cai-Jin Chen, Yu-Tse Tsan, Po-Yu Liu, Yu-Wei Chan, and Wei-Chen Chan. An implementation of real-time air quality and influenza-like illness data storage and processing platform. *Computers in Human Behavior*, 2018.

[15] T. Ram Prakash, Misha Kakkar, and Kritika Patel. Geo-identification of web users through logs using elk stack. *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, pages 606–610, 2016.

[16] James Zhang, Robert Gardner, and Ilija Vukotic. Anomaly detection in wide area network meshes using two machine learning algorithms. *Future Generation Computer Systems*, 93:418 – 426, 2019.

[17] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, and Haojin Zhu. Hardening malware detection systems against cyber maneuvers: An adversarial machine learning approach. *CoRR*, abs/1706.04146, 2017.

[18] Patrick Peterson. Unmasking deceptive attacks with machine learning. *Computer Fraud & Security*, 2018(11):15 – 17, 2018.

[19] Muhammad Al-Qurishi, Majed Alrubaian, Sk Md Mizanur Rahman, Atif Alamri, and Mohammad Mehedi Hassan. A prediction system of sybil attack in social network using deep-regression model. *Future Generation Computer Systems*, 87:743 – 753, 2018.

[20] Hongyu Liu, Bo Lang, Ming Liu, and Hanbing Yan. Cnn and rnn based payload classification methods for attack detection. *Knowledge-Based Systems*, 163:332 – 341, 2019.

[21] P. P. I. Langi, , W. Najib, and T. B. Aji. An evaluation of twitter river and logstash performances as elasticsearch inputs for social media analysis of twitter. In *2015 International Conference on Information Communication Technology and Systems (ICTS)*, pages 181–186, Sep. 2015.

[22] Ibrahim Ghafir, Mohammad Hammoudeh, Vaclav Prenosil, Liangxiu Han, Robert Hegarty, Khaled Rabie, and Francisco J. Aparicio-Navarro. Detection of advanced persistent threat using machine-learning correlation analysis. *Future Generation Computer Systems*, 89:349 – 359, 2018.

[23] Abebe Abeshu Diro and Naveen Chilamkurti. Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, 82:761 – 768, 2018.

[24] S Bagnasco, D Berzano, A Guarise, S Lusso, M Masera, and S Vallero. Monitoring of IaaS and scientific applications on the cloud using the elasticsearch ecosystem. *Journal of Physics: Conference Series*, 608:012016, may 2015.

[25] Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir, and Banu Diri. Machine learning based phishing detection from urls. *Expert Systems with Applications*, 117:345 – 357, 2019.

[26] Rafał Kozik, Michał Choraś, Massimo Ficco, and Francesco Palmieri. A scalable distributed machine learning approach for attack detection in edge computing environments. *Journal of Parallel and Distributed Computing*, 119:18 – 26, 2018.

[27] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. An evaluation framework for network security visualizations. *Computers & Security*, 84:70 – 92, 2019.

[28] Zhuo Chen, Fu Jiang, Yijun Cheng, Xin Gu, Weirong Liu, and Jun Peng. Xgboost classifier for ddos attack detection and analysis in sdn-based cloud. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 251–256. IEEE, 2018.

[29] X. Yuan, C. Li, and X. Li. Deepdefense: Identifying ddos attack via deep learning. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8, May 2017.

[30] Peiyuan Sun, Jianxin Li, Md Zakirul Alam Bhuiyan, Lihong Wang, and Bo Li. Modeling and clustering attacker activities in iot through machine learning techniques. *Information Sciences*, 479:456 – 471, 2019.

Appendix A

ELK Stack Installation

I. Installation front package

```
# sudo apt-get update
# sudo apt-get install -y vim ntp curl ssh
```

II. install openjdk-8-jdk

```
# sudo apt-get install -y openjdk-8-jdk
# sudo ln -s /usr/lib/jvm/java-8-openjdk-amd64 /usr/lib/jvm/jdk
```

III. Add the environment variable

```
# sudo vim .bashrc
export JAVA_HOME=/usr/lib/jvm/jdk/
# source .bashrc
```

IV. Authentication key

```
 # wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
```

V. Install package

```
# sudo apt-get install apt-transport-https ca-certificates -y
# echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main"| sudo tee -a /etc/apt/sources.l
# sudo apt-get update
```

VI. Install Elasticsearch

```
# sudo apt-get install -y elasticsearch
```

VII. Modify Elasticsearch.yml

```
# sudo vim /etc/elasticsearch/elasticsearch.yml

network.host: IP address
http.port: 9200
```

VIII. Start Elasticsearch

```
# sudo systemctl start elasticsearch
```

IX. Install Logstash

```
# sudo apt-get install logstash
```

X. Start Logstash

```
# sudo systemctl start logstash
```

XI. Install Kibana

```
# sudo apt-get install kibana
```

XII. Modify Kibana.yml

```
# sudo vim /etc/kibana/kibana.yml
server.port:5601
server.host: "IP address"
elasticsearch.url: "http://IP address:9200"
```

XIII. Start Kibana

```
# sudo systemctl start kibana
```

XIII. ELK Stack monitoring website

```
# http://IP address:5601
```

Appendix B

Data processing

I. Load and pyplot dataset

```
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot

series = read_csv('nfcapd.201901091445.txt', engine='python', skipfooter=4, header=None, skiprows=[0])
series.columns = ['Date first seen', 'Date last seen', 'Duration', 'Src IP Addr', 'Dst IP Addr', 'Src
import pandas as pd
series.to_csv('out1.csv', encoding="utf-8-sig")
```

II. Data processing

```
import re
import pandas
dataset = pandas.read_csv('out1.csv', engine='python', skipfooter=4) #, usecols=[1,6]
#dataset.set_index ("Date first seen", inplace=True)
#dataset.index = pd.DatetimeIndex(dataset.index)
#dataset.sort_values('Date first seen', inplace=True)
del dataset["嚜 "],dataset["Flags"],dataset["Out Pkt"],dataset["Out Byte"]#,dataset["Date first seen"]
dataset

if dataset['In Byte'].dtype != 'int64':
dataset['In Byte']=dataset['In Byte'].str.lstrip()
dataset['In Byte']=dataset['In Byte'].str.rstrip()
InByte = dataset['In Byte'].str.strip().str.split('.',0,True)
```

```
InByte.columns = ['0', '1']
InByte = InByte.fillna('.0')
InByte['1'] = InByte['1'].replace(' M', '00.0', regex=True)
InByte['1'] = InByte['1'].replace(' G', '00000.0', regex=True)
dataset['In Byte'] = InByte['0']+InByte['1']
InByte = dataset['In Byte'].str.strip().str.split('.',0,True)
dataset['In Byte'] = InByte[0]
dataset['In Byte'] = dataset['In Byte'].astype(int)

if dataset['In Pkt'].dtype != 'int64':
dataset['In Pkt']=dataset['In Pkt'].str.lstrip()
dataset['In Pkt']=dataset['In Pkt'].str.rstrip()
InPkt = dataset['In Pkt'].str.strip().str.split('.',0,True)
InPkt.columns = ['0', '1']
InPkt = InPkt.fillna('.0')
InPkt['1'] = InPkt['1'].replace(' M', '00.0', regex=True)
InPkt['1'] = InPkt['1'].replace(' G', '00000.0', regex=True)
dataset['In Pkt'] = InPkt['0']+InPkt['1']
InPkt = dataset['In Pkt'].str.strip().str.split('.',0,True)
dataset['In Pkt'] = InPkt[0]
dataset['In Pkt'] = dataset['In Pkt'].astype(int)


dataset['Dst Pt'] = dataset['Dst Pt'].astype(int)
dataset

dataflow = dataset
from sklearn.preprocessing import LabelEncoder
dataflow['Proto'] = dataflow['Proto'].str.lstrip()
dataflow['Proto'] = dataflow['Proto'].str.rstrip()
le = LabelEncoder()

le.fit(['ICMP','IGMP', 'TCP','UDP'])

dataflow['Proto'] = le.transform(dataflow['Proto'])
le.transform(['ICMP','IGMP','TCP','UDP'])

dataflow['Src IP Addr'] = dataflow['Src IP Addr'].str.lstrip()
dataflow['Src IP Addr'] = dataflow['Src IP Addr'].str.rstrip()
dataflow['Dst IP Addr'] = dataflow['Dst IP Addr'].str.lstrip()
dataflow['Dst IP Addr'] = dataflow['Dst IP Addr'].str.rstrip()
```

III. Attack data analysis

```
CodeRed = dataflow.loc[(dataflow['Dst Pt'].astype(int) == 80 ) & (dataflow['In Pkt'].astype(int)==3) &
CodeRed['attack']='attack'
```

```
if CodeRed.shape[0] > 1:
print(CodeRed['Src IP Addr'].drop_duplicates(keep='first'))
else:
print("no CodeRed attack")

Nimda = dataflow.loc[(dataflow['Dst Pt'].astype(int) == 80 ) ]
from collections import Counter
A = Nimda['Src IP Addr'].value_counts() >=1000
A = Counter(A)[1]
if A !=0 :
for i in range (A):
Nimda_IP=Nimda['Src IP Addr']
Cou_IP=Counter(Nimda_IP)
Top_IP = Cou_IP.most_common()[i][0]
Top_IP_Cou = Cou_IP.most_common()[i][1]
if (Top_IP_Cou >=1000):
Nimda_A = Nimda.loc[(Nimda['Src IP Addr'] == Top_IP )]
Nimda_A['attack']='attack'
print(Top_IP + "對外連線次數為%d" % (Top_IP_Cou))
else:
Nimda_A=None
print('no Nimda attack')
else:
Nimda_A=None
print('no Nimda attack')

from collections import Counter
Worm = dataflow.loc[(dataflow['Dst Pt'].astype(int) == 445 ) & (dataflow['Proto'] == 2)]
A = Worm['Src IP Addr'].value_counts() >=1000
A = Counter(A)[1]
Worm_A = None
if A !=0 :
for i in range (0,A):
Worm_IP=Worm['Src IP Addr']
Worm_Cou_IP=Counter(Worm_IP)
Worm_Top_IP = Worm_Cou_IP.most_common()[i][0]
Worm_Top_IP_cou = Worm_Cou_IP.most_common()[i][1]
Worm_A = pd.concat([Worm_A, dataflow.loc[(dataflow['Dst Pt'].astype(int) == 445 ) & (dataflow['Proto']
Worm_A['attack']='attack'
if Worm_Top_IP_cou > 1000 :
print ("IP: " + Worm_Top_IP + "在Port445的TCP連接次數為%d" % (Worm_Top_IP_cou))

else:
print("no Worm attack")
else:
print("no Worm attack")

attack= pd.concat([CodeRed, Nimda_A, Worm_A])
```

IV. Merge multiple attack data

```
all_attack= pd.concat([attack1, attack2, attack3, attack4, attack5, attack6,attack7 ,attack8 ,attack9
all_attack
```

V. None attack data

```
series = read_csv('nfcapd.201901091700.txt', engine='python', skipfooter=4, header=None, skiprows=[0])
series.columns = ['Date first seen', 'Date last seen', 'Duration', 'Src IP Addr', 'Dst IP Addr', 'Src
series.to_csv('out13.csv', encoding="utf-8-sig")
dataset = pandas.read_csv('out13.csv', engine='python', skipfooter=4)
del dataset["噩 "],dataset["Flags"],dataset["Out Pkt"],dataset["Out Byte"]#,dataset["Date first seen"]
if dataset['In Byte'].dtype != 'int64':
dataset['In Byte']=dataset['In Byte'].str.lstrip()
dataset['In Byte']=dataset['In Byte'].str.rstrip()
InByte = dataset['In Byte'].str.strip().str.split('.',0,True)
InByte.columns = ['0', '1']
InByte = InByte.fillna('.0')
InByte['1'] = InByte['1'].replace(' M', '00.0', regex=True)
InByte['1'] = InByte['1'].replace(' G', '00000.0', regex=True)
dataset['In Byte'] = InByte['0']+InByte['1']
InByte = dataset['In Byte'].str.strip().str.split('.',0,True)
dataset['In Byte'] = InByte[0]
dataset['In Byte'] = dataset['In Byte'].astype(int)
if dataset['In Pkt'].dtype != 'int64':
dataset['In Pkt']=dataset['In Pkt'].str.lstrip()
dataset['In Pkt']=dataset['In Pkt'].str.rstrip()
InPkt = dataset['In Pkt'].str.strip().str.split('.',0,True)
InPkt.columns = ['0', '1']
InPkt = InPkt.fillna('.0')
InPkt['1'] = InPkt['1'].replace(' M', '00.0', regex=True)
InPkt['1'] = InPkt['1'].replace(' G', '00000.0', regex=True)
dataset['In Pkt'] = InPkt['0']+InPkt['1']
InPkt = dataset['In Pkt'].str.strip().str.split('.',0,True)
dataset['In Pkt'] = InPkt[0]
dataset['In Pkt'] = dataset['In Pkt'].astype(int)
dataset['Dst Pt'] = dataset['Dst Pt'].astype(int)
dataflow = dataset
from sklearn.preprocessing import LabelEncoder
dataflow['Proto'] = dataflow['Proto'].str.lstrip()
dataflow['Proto'] = dataflow['Proto'].str.rstrip()
le = LabelEncoder()
```

```
le.fit(['ICMP','IGMP', 'TCP','UDP'])

dataflow['Proto'] = le.transform(dataflow['Proto'])
le.transform(['ICMP','IGMP','TCP','UDP'])
dataflow['Src IP Addr'] = dataflow['Src IP Addr'].str.lstrip()
dataflow['Src IP Addr'] = dataflow['Src IP Addr'].str.rstrip()
dataflow['Dst IP Addr'] = dataflow['Dst IP Addr'].str.lstrip()
dataflow['Dst IP Addr'] = dataflow['Dst IP Addr'].str.rstrip()
#CodeRed檢測
CodeRed_13 = dataflow.loc[(dataflow['Dst Pt'].astype(int) == 80 ) & (dataflow['In Pkt'].astype(int)==3
CodeRed_13['attack']='attack'
if CodeRed_13.shape[0] > 1:
print(CodeRed_13['Src IP Addr'].drop_duplicates(keep='first'))
else:
print("no CodeRed attack")

#Nimda檢測
Nimda = dataflow.loc[(dataflow['Dst Pt'].astype(int) == 80 ) ]
from collections import Counter
A = Nimda['Src IP Addr'].value_counts() >=1000
A = Counter(A)[1]
if A !=0 :
for i in range (A):
Nimda_IP=Nimda['Src IP Addr']
Cou_IP=Counter(Nimda_IP)
Top_IP = Cou_IP.most_common()[i][0]
Top_IP_Cou = Cou_IP.most_common()[i][1]
if (Top_IP_Cou >=1000):
Nimda_13 = Nimda.loc[(Nimda['Src IP Addr'] == Top_IP )]
Nimda_13['attack']='attack'
print(Top_IP + "對外連線次數為%d" % (Top_IP_Cou))
else:
Nimda_13=None
print('no Nimda attack')
else:
Nimda_13=None
print('no Nimda attack')

#震荡波病毒分析
from collections import Counter
Worm = dataflow.loc[(dataflow['Dst Pt'].astype(int) == 445 ) & (dataflow['Proto'] == 2)]
A = Worm['Src IP Addr'].value_counts() >=1000
A = Counter(A)[1]
Worm_13 = None
if A !=0 :
for i in range (0,A):
Worm_IP=Worm['Src IP Addr']
Worm_Cou_IP=Counter(Worm_IP)
```

```
Worm_Top_IP = Worm_Cou_IP.most_common()[i][0]
Worm_Top_IP_cou = Worm_Cou_IP.most_common()[i][1]
Worm_13 = pd.concat([Worm_13, dataflow.loc[(dataflow['Dst Pt'].astype(int) == 445 ) & (dataflow['Proto
Worm_13['attack']='attack'
if Worm_Top_IP_cou > 1000 :
print ("IP: " + Worm_Top_IP + "在Port445的TCP連接次數為%d" % (Worm_Top_IP_cou))

else:
print("no Worm attack")
else:
print("no Worm attack")

Normal_data = pd.concat([dataflow, CodeRed_13, Nimda_13, Worm_13])
Normal_data = Normal_data.drop_duplicates(subset = None, keep = False, inplace=False)
Normal_data['attack']='None attack'
Normal_data = Normal_data[0:114000]
Normal_data
```

VI. Concat normal and attack data

```
all_data= pd.concat([Normal_data, all_attack])
all_data
```

VII. LabelEncoder

```
all_data['Src IP Addr'] = all_data['Src IP Addr'].str.lstrip()
all_data['Dst IP Addr'] = all_data['Dst IP Addr'].str.rstrip()

le = LabelEncoder()

le.fit(all_data['Dst IP Addr'])
all_data['Dst IP Addr'] = le.transform(all_data['Dst IP Addr'])
le.fit(all_data['Src IP Addr'])
all_data['Src IP Addr'] = le.transform(all_data['Src IP Addr'])

le.fit(all_data['attack'])
all_data['attack'] = le.transform(all_data['attack'])
```

VIII. Data to csv

```
all_data.to_csv('all_data.csv')
```

Appendix C

XGBoost Machine model

I. Import XGBoost packages

```
import os
os.environ["PATH"] += os.pathsep + 'C:\ProgramData\Anaconda3\Library\bin'
import numpy as np
import xgboost as xgb
import pandas as pd
from pandas import read_csv
from xgboost import XGBClassifier
from xgboost import plot_importance
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
\noindent II. Unzip hbase-1.0.0-hadoop2-bin.tar.gz\\
\begin{lstlisting} [frame=single]
# tar zxf hbase-1.0.0-hadoop2-bin.tar.gz
```

II. Data spilt

```
np.random.seed(7)
dataset = read_csv("all_data.csv")
del dataset["Unnamed: 0"],dataset["Date first seen"],dataset["Date last seen"],dataset['Src IP Addr'],
#,dataset["Input"],dataset["Output"],dataset['Src IP Addr'],dataset['Dst IP Addr']

# split data into X and Y
X = dataset[['Duration','Src Pt','Dst Pt','Proto','In Pkt','In Byte','Input','Output']].values #特征集
```

```
#'Date first seen','Date last seen','Input','Output','Src IP Addr','Dst IP Addr'
Y = dataset[['attack']].values#標籤集
# split data into train and test sets  # 拆分数据集
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=7)
feature_names = ['Duration','Src Pt','Dst Pt','Proto','In Pkt','In Byte','Input','Output']
#'Duration','Src IP Addr','Dst IP Addr','Input','Output'
dtrain = xgb.DMatrix(X_train, y_train, feature_names = feature_names)
dtest = xgb.DMatrix(X_test, y_test, feature_names = feature_names)

dataset.info()
```

III. XGBoost model

```
model = xgb.XGBClassifier(learning_rate = 0.008597735392699073, objective = 'binary:logistic',
n_estimators = 472, n_jobs = 4,
max_depth = 4, min_child_weight = 8,
seed = 30, max_delta_step = 6,
subsample = 0.7065226970496071, colsample_bytree = 0.515980616951696,
gamma = 0.044644589297517134, scale_pos_weight = 46,#1
reg_alpha = 80, reg_lambda = 38)
```

IV. XGBoost model training and test

```
eval_set = [(X_train, y_train),(X_test, y_test)]
model.fit(X_train, y_train,
early_stopping_rounds=49,
eval_metric=["rmse","auc","error", "logloss"],
eval_set=eval_set,
verbose=True)
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

V. save XGBoost model

```
model.save_model('001.model')
```

VI. XGBoost model attack val

```
np.random.seed(7)
dataset = read_csv("allattack.csv")
del dataset['Unnamed: 0'],dataset['Date first seen'],dataset['Date last seen'],dataset['Src IP Addr'],
#,dataset['Src IP Addr'],dataset['Dst IP Addr'],dataset['Input'],dataset['Output']
X_val = dataset[['Duration','Src Pt','Dst Pt','Proto','In Pkt','In Byte','Input','Output']]
Y_val = dataset[['attack']]
X_val.to_csv('X_val')
Y_val.to_csv('Y_val')
feature_names = ['Duration','Src Pt','Dst Pt','Proto','In Pkt','In Byte','Input','Output']
#,'Src IP Addr','Dst IP Addr','Input','Output'
# split data into X and Y
X_val = dataset[['Duration','Src Pt','Dst Pt','Proto','In Pkt','In Byte','Input','Output']].values #特
#'Date first seen','Date last seen','Src IP Addr','Dst IP Addr','Input','Output'
Y_val = dataset[['attack']].values#標籤集
attack_x = xgb.DMatrix(X_val, feature_names = feature_names)
#attack_y = xgb.DMatrix(Y_val)
#Val_attack = xgb.DMatrix(Y_val)#, feature_names = feature_names)
bst = xgb.Booster(model_file='001.model')

#assert np.sum(np.abs(preds - y_pred)) == 0
preds = bst.predict(attack_x)
predictions = [round(value) for value in preds]

accuracy = accuracy_score(Y_val, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

import numpy as np
predictions = model.predict(X_val)
np.set_printoptions(threshold=np.inf)
d = np.rint(predictions)
from collections import Counter
def row_counter(d):
lit_of_tups = [tuple(ele) for ele in d]
return Counter(lit_of_tups)
row_counter(d)
print("異常數據量:", (row_counter(d)[(1.0,)]))
```

VI. XGBoost model new raw val

```
dataset = read_csv("attack_01val.csv")
del dataset['Unnamed: 0'],dataset['Date first seen'],dataset['Date last seen'],dataset['Src IP Addr'],
#,dataset['Src IP Addr'],dataset['Dst IP Addr'],dataset['Input'],dataset['Output']
#X_val = dataset[['Duration','Src Pt','Dst Pt','Proto','In Pkt','In Byte','Input','Output']]
#Y_val = dataset[['attack']]
#X_val.to_csv('X_val')
```

```
#Y_val.to_csv('Y_val')
feature_names = ['Duration','Src Pt','Dst Pt','Proto','In Pkt','In Byte','Input','Output']
#,'Src IP Addr','Dst IP Addr','Input','Output'
# split data into X and Y
X_01val = dataset[['Duration','Src Pt','Dst Pt','Proto','In Pkt','In Byte','Input','Output']].values #
#'Date first seen','Date last seen','Src IP Addr','Dst IP Addr','Input','Output'
Y_01val = dataset[['attack']].values#標籤集
attack_x = xgb.DMatrix(X_01val, feature_names = feature_names)
#attack_y = xgb.DMatrix(Y_val)
#Val_attack = xgb.DMatrix(Y_val)#, feature_names = feature_names)
bst = xgb.Booster(model_file='001.model')

#assert np.sum(np.abs(preds - y_pred)) == 0
preds = bst.predict(attack_x)
predictions = [round(value) for value in preds]

accuracy = accuracy_score(Y_01val, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

import numpy as np
predictions = model.predict(X_val)
np.set_printoptions(threshold=np.inf)
d = np.rint(predictions)
from collections import Counter
def row_counter(d):
lit_of_tups = [tuple(ele) for ele in d]
return Counter(lit_of_tups)
row_counter(d)
print("異常數據量:", (row_counter(d)[(1.0,)]))
```

Appendix D

DNN model

I. Import keras packages

```
import numpy as np
from pandas import read_csv
from matplotlib import pyplot as plt
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from keras import optimizers
from keras.optimizers import Adadelta
```

II. Read training csv data and spilt

```
filename = 'BB_all_data.csv'
footer = 1
data = read_csv(filename, header=None, engine='python', skiprows=footer)
#data = read_csv(filename)
del data[0], data[1], data[2], data[5]
#, data[8], data[9], data[3], data[11]
data = data.values
X = data[:,1:8]
Y = data[:,9]
#Y = A
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.4, random_state=8)
```

III. DNN model construction and training

```
model = Sequential()
model.add(Dense(256, input_dim=7, kernel_initializer='uniform',  activation='softplus'))
model.add(Dropout(0.3))
model.add(Dense(256, kernel_initializer='uniform',  activation='softplus'))
model.add(Dropout(0.5))
model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
model.summary()
optimizer=Adadelta(lr=0.001)
model.compile(optimizer=optimizer , loss='binary_crossentropy', metrics=['accuracy'])
# Fit the model
history = model.fit(X_train, y_train, epochs=50, batch_size=100, verbose=2, validation_data=(X_test, y
```

IV. Accuracy of the test

```
predictions = model.predict(X_test)
import matplotlib.pyplot as plt
history_dict= history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(loss_values)+1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.axis([0, 30, 0.0, 4.5])
plt.show()

import matplotlib.pyplot as plt
history_dict= history.history
acc = history_dict['acc']
val_acc = history_dict['val_acc']
epochs = range(1,len(loss_values)+1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation acc')
plt.xlabel('Epochs')
plt.ylabel('acc')
plt.legend()
```

```
plt.axis([0, 30, 0.3, 1.0])
plt.show()

score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Accuracy: %.2f%%" % (score[1] * 100.0))
```

V. model normal val predictions

```
filename = 'attack_01val.csv'
footer = 1

data = read_csv(filename, header=None, engine='python', skiprows=footer)
del data[0], data[1], data[2], data[3]
#, data[8], data[9], data[3], data[11]
data
data = data.values
X = data[:,1:8]
Y = data[:,9]
from keras.models import load_model
model = load_model('BB_DL_model')
from sklearn.metrics import accuracy_score
score, acc = vmodel.evaluate(X, Y,batch_size=128)
print('Test score:', score)
print('Test accuracy:', acc)
print("Accuracy: %.2f%%" % (acc * 100.0))

import numpy as np
predictions = model.predict(X_v)
np.set_printoptions(threshold=np.inf)
d = np.rint(predictions)
from collections import Counter
def row_counter(d):
lit_of_tups = [tuple(ele) for ele in d]
return Counter(lit_of_tups)
row_counter(d)
print("異常數據量:", (row_counter(d)[(1.0,)]))
```

Appendix E

RNN model

I. Import keras packages

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, LSTM, TimeDistributed, RepeatVector
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from sklearn.model_selection import train_test_split
from keras.layers import Input, SimpleRNN, Activation
from keras import optimizers
from keras.optimizers import RMSprop
```

II. Data spilt

```
filename = 'BB_all_data.csv'
footer = 1
data = read_csv(filename, header=None, engine='python', skiprows=footer)
```

```
#data = read_csv(filename)
del data[0], data[1], data[2], data[5]
data = data.values
X = data[:,1:8]
Y = data[:, 9]
#Y = A
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.4, random_state=8)
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

III. RNN model construction

```
# sudo vim ~/.bashrc
model = Sequential()
model.add(SimpleRNN(256, input_dim=7, kernel_initializer='uniform',  activation='softplus', return_seq
model.add(Dropout(0.3))
model.add(SimpleRNN(256, input_dim=7, kernel_initializer='uniform',  activation='softplus', return_seq
model.add(Dropout(0.5))
model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))

model.summary()

from keras import optimizers
from keras.optimizers import Adadelta

optimizer=Adadelta(lr=0.001)
model.compile(optimizer=optimizer , loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=50, batch_size=100, verbose=2, validation_data=(X_test, y
predictions = model.predict(X_test)
print(predictions)
```

IV. RNN model test accuracy

```
import matplotlib.pyplot as plt
history_dict= history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1,len(loss_values)+1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()
plt.axis([0, 50, 0.0, 3.5])
plt.show()

import matplotlib.pyplot as plt
history_dict= history.history
acc = history_dict['acc']
val_acc = history_dict['val_acc']

epochs = range(1,len(loss_values)+1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation acc')
plt.xlabel('Epochs')
plt.ylabel('acc')
plt.legend()
plt.axis([0, 50, 0.3, 1.00])
plt.show()

score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Accuracy: %.2f%%" % (score[1] * 100.0))
```

V. Save RNN model')

```
model.save('attack_rnn_model')
```

VI. RNN model attack data accuracy

```
filename = 'allattack.csv'
footer = 1

vdata = read_csv(filename, header=None, engine='python', skiprows=footer)
#data = read_csv(filename)

del vdata[0], vdata[1], vdata[2], vdata[3]
#, data[8], data[9], data[3], data[11]
from keras.models import load_model
vmodel = load_model('attack_rnn_model')
vdata = vdata.values
X_v = vdata[:,1:8]
Y_v = vdata[:,9]
```

```
#Y_v = B
X_v = np.reshape(X_v, (X_v.shape[0], 1, X_v.shape[1]))
from sklearn.metrics import accuracy_score
score, acc = vmodel.evaluate(X_v, Y_v,batch_size=128)
print('Test score:', score)
print('Test accuracy:', acc)
print("Accuracy: %.2f%%" % (acc * 100.0))
import numpy as np

predictions = model.predict(X_v)
np.set_printoptions(threshold=np.inf)
d = np.rint(predictions)

from collections import Counter
def row_counter(d):
lit_of_tups = [tuple(ele) for ele in d]
return Counter(lit_of_tups)
row_counter(d)
print("異常數據量:", (row_counter(d)[(1.0,)]))
```