國 立 成 功 大 學
資 訊 工 程 學 研 究 所
碩 士 論 文

基於機器學習和深度學習偵測物聯網殭屍網路散佈的惡意軟體

Detecting malware distributed by IoT Botnet based on Machine Learning and Deep Learning

研 究 生：黃志翔
指 導 教 授： 張燕光 博士

中 華 民 國 一 一 二 年 一 月

國立成功大學

碩士論文

基於機器學習和深度學習偵測物聯網殭屍網路散佈的惡意軟體

Detecting malware distributed by IoT Botnet based on Machine Learning and Deep Learning

研究生：黃志翔

本論文業經審查及口試合格特此證明

論文考試委員：張基宏

陳育業

王瑋

指導教授：張基宏

單位主管：張基宏

（單位主管是否簽章授權由各院、系（所、學位學程）自訂）

中 華 民 國 112 年 1 月 6 日

黃志翔*, 張燕光**

國立成功大學資訊工程研究所

摘要

自從 COVID-19 疫情爆發以來，許多政府部分僅能利用網路的方式處理民眾的相關事務，大量民眾的個資都在網路上流傳，網路犯罪者看到這樣的趨勢，許多物聯網設備也因此被網路犯罪者入侵，大量和無處不在的物聯網設備吸引了網路犯罪者進行網路攻擊和偷取個資，因此物聯網資訊安全成為現今巨大的挑戰之一。越來越多的物聯網惡意軟體嚴重威脅到個資還有一些昂貴的物聯網設備。許多物聯網設備的作業系統都是 Linux，因此在這篇論文中，我們收集了許多惡意的可執行可鏈接文件。我們利用三種方法分析這些可執行可鏈接文件，第一個方法是將這些可執行可鏈接文件經過 limon sandbox 分析產生報告後，用 python 程式語言將所有特徵值整理成一個 csv 檔案，然後最後作機器學習和深度學習分析，這一個方法也就是所謂的動態分析。第二個方法是直接將這些可執行可鏈接文件經過 Linux 內建的指令分析產生結果，例如 strace，也是用 python 程式語言將所有特徵值整理成一個 csv 檔案，然後最後作機器學習和深度學習分析，這一個方法也就是所謂的靜態分析。第三個個方法則是直接將這些可執行可鏈接文件轉換成二維灰階圖片，然後最後作深度學習分析。

最後也有討論到不是全部的可執行可鏈接文件的結果報告都可以完整呈現出來，有一些惡性的可執行可鏈接文件會不斷的在迴圈內重複，造成無法在結果報告中呈現出來，而在我們的資料集當中有 46 筆惡性的可執行可鏈接文件在第一個方法無法完整呈現，因此在本篇論文中我們衍生出後面兩種方法，試著能夠檢測所有可執行可鏈接文件。在結論可以得知每個分析方法都各有優缺點，決策樹和隨機森林的準確率在三個方法中都是最高的，高斯貝氏的準確率在三個方法中則是最低的。

*作者　**指導教授

關鍵字:殭屍網路、機器學習、深度學習、物聯網

Chih Hsiang Huang*, and Yeim Kuan Chang**

Institute of Computer Science and Information Engineering,

National Cheng Kung University, Tainan, Taiwan, R.O.C.

Abstract

Since the outbreak of COVID-19, many government departments can only use the Internet to deal with people's related affairs, and a large number of people's personal data are circulated on the Internet. Cybercriminals have seen this trend, and many IoT devices have also Therefore hacked by cybercriminals, the large number and ubiquity of IoT devices attract cybercriminals to carry out cyber attacks and steal personal data. IoT information security has become one of the biggest challenges today. More and more IoT malware is a serious threat to personal data as well as some expensive IoT devices. The operating system of many IoT devices is Linux, so in this paper, we collected many malicious executable linkable files. We use three methods to analyze these executable and linkable files. The first method is to analyze these executable and linkable files through limon sandbox to generate a report, and then use the python programming language to organize all the feature values into a csv file, and finally make a machine Learning and deep learning analysis, this method is also called dynamic analysis. The second method is to directly analyze these executable and linkable files through the built-in instructions of Linux to generate results, such as strace. It also uses the python programming language to organize all the feature values into a csv file, and then finally do machine learning and deep learning analysis. This method is also called static analysis. The third method is to directly convert these executable and linkable

files into two-dimensional grayscale images, and then perform deep learning analysis at the end.

Finally, it was also discussed that not all executable and linkable files can be fully presented in the result report, and some vicious executable and linkable files will be repeated in the loop, resulting in the failure to be presented in the result report. Our dataset has 46 malicious executable and linkable files cannot be fully presented by the first method, so in this paper we derive the latter two methods to try to detect all executable and linkable files. In the conclusion, we can know that each analysis method has its own advantages and disadvantages. The accuracy rate of decision tree and random forest is the highest among the three methods, and the accuracy rate of Gaussian Bayesian is the lowest among the three methods.

* Author ** Advisor

Keywords: Botnets, Machine learning, Deep learning, Internet of Things (IoT)

誌謝

在碩士的這兩年半中，碩一修了很多精實的課程，也過得非常充實，感謝每門課老師認真地教學，在完成學士班學位後，先當完兵休息一年後，最後決定再回來讀碩士學位，這篇論文的完成以及口試的結束，要感謝這兩年半幫助的老師．

首先非常的感謝我的指導教授 張燕光教授，能夠耐心的指導學生，在研究過程中，經常會有對於計算機網路不懂的部分，老師依然會不厭其煩的解說以及給予了很多建議與指導，平時也會分享如何才能夠有高效率的學習方式以及生活上的有趣的事情，如此一來，我才能夠得到如此充實的生活，也讓我不會在研究上迷失方向．在論文撰寫過程中，也給予了很多建議以及需要注意的事項，也非常感謝教授能夠在初稿的完成後，詳細的閱讀過我的論文並且給予糾正．

最後謝謝實驗室的同學，讓我在學習上遇到問題時有人可以詢問，遇到困難時我們能夠一起想辦法解決，在平常課業上的探討，讓我的學習不會變得枯燥乏味，也非常感謝在口試時的幫助，讓整個口試流程能夠順利圓滿的結束。

黃志翔 謹誌於

國立成功大學 資訊工程研究所

中華民國 112 年 2 月

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

Software that is classified as malware is primarily based on the intent of its developers rather than its actual functionality. Malware continues to thrive as new types of crime occur every day, and organized cybercriminals often reap huge profits. Malware was born out of experimentation and shenanigans, but eventually evolved to wreak havoc on targeted machines. Nowadays, there are many types of malwares, the term malware, refers to a program used to damage a computer, which may be spyware, keyloggers, viruses, worms, Trojan horses, or any malicious code intended to infiltrate a computer.

With the vigorous development and rapid popularization of network technology, people can achieve instant and zero-distance interaction through the network. In recent years, devices that can be connected to the Internet have sprung up, the vision of an Internet of everything and people can interact with it through the Internet has become more and more clear, and a revolution initiated by the terminal has also been born.

In recent years, the field of artificial intelligence has made some progress in using data to build models to solve practical problems. Major advances, in terms of the amount of data required, can be roughly divided into two categories: machine learning that uses a small amount of data and deep learning that requires a large amount of data. When more new data are obtained, the enterprise can use these data to recalculate to obtain a more accurate and effective new model for deployment on the system. The wireless network technologies and protocols of various sensing devices and various connection devices that constitute the Internet of Things, together with the back-end edge computing servers and cloud environment servers connected through the Internet, form a real-time access to a large number of various types of the ecological structure of the data, which just constitutes a large

amount of data sources required by the deep learning technology, creates the synergy of the Internet of Things and artificial intelligence technology, and dynamically deploys the big data analysis model generated by artificial intelligence to various types of data under the Internet of Things architecture. On the application system, create new application solutions. The rise and development of big data, artificial intelligence and the Internet of Things have brought about innovative technological applications. The artificial intelligence system is introduced into the Internet of Things technology, and it becomes the Artificial Intelligence of Things. Combined with artificial intelligence, IoT obtains information from data and continuously evolves through data accumulation. It can learn intelligently and provide the best experience of customized services.

Information security issues arising from Internet of Things devices must be taken seriously, because compared with computers, these devices may touch more personal privacy, or affect personal safety and national security. Once attacked by hackers, the consequences will be disastrous. The security of IoT devices is closely related to us. Whether you are a manufacturer or a user, we must understand how to reduce their information security risks. However, according to a survey report conducted by AT&T in 2015, as many as 85% of the 5,000 companies interviewed want to develop IoT applications, but only 10% of the companies are confident that they can cope with hacker attacks. [2] Even enterprises generally are not sure about the information security issues of the Internet of Things.

Another aspect is that IoT devices will bring serious security risks and network attacks. Take the large-scale DDoS attack on DNS provider Dyn as an example, part of which was launched by the IoT device botnet controlled by Mirai. This incident has affected well-known websites that use Dyn services, including the BBC and GitHub. However, the way for hackers to gain control over these IoT devices is only by testing 60 common preset account numbers and passwords.

In the manufacturing industry, we have seen many lighthouse factories driving Industry 4.0 through the Internet of Things and achieving excellent results. Others are cities, retail, home, and

energy are gradually moving towards wisdom through the Internet of Things.

Although these concepts may seem out of reach, leaders across industries are constantly moving in this direction. According to Gartner, by 2020, there will be 20 billion interconnected devices around the world, and half of them will be online. The Internet of Things will fundamentally transform the way business operates, making entire organizations more data-driven.

The term botnet is used to refer to a "collection of soft robots". The word is usually associated with malware, but it can also be used to refer to a "computer network using distributed computing software". A botnet is often named after its malware name, and there are many botnets that operate using the same malware family but are operated by different perpetrators. Botnet can be used to refer to any group of bots like IRC bots but the word is usually used to refer to a group of zombies that execute software, and the software usually exploit the weakness of the browser. Under a command-and-control framework, users can install files when they download files, such as worms, Trojan horses or backdoors.

With the development of Internet of Things (IoT) technology, hackers' attack targets are no longer limited to computers or mobile phones. And all kinds of things around us can be seen everywhere combined with IoT, whether it is work environment, campus, enterprise, transportation, furniture or even medical equipment, etc., are closely related to the Internet.

By discussing the security and privacy of the Internet of Things, we learned that although IoT life brings many conveniences and benefits, if there is no perfect information security consideration, it will bring serious information security risks. Many devices that were thought not to be hacked have become one of the targets of hackers, such as monitors, smart cars, smart doors, etc.

In terms of information security considerations, IoT devices, compared with computers, lack a complete protection mechanism. As long as there is a security weakness on the device, hackers can easily implant malicious software through the loopholes and take them under their command for malicious behavior. For example, in the information security network incident that occurred in 2016,

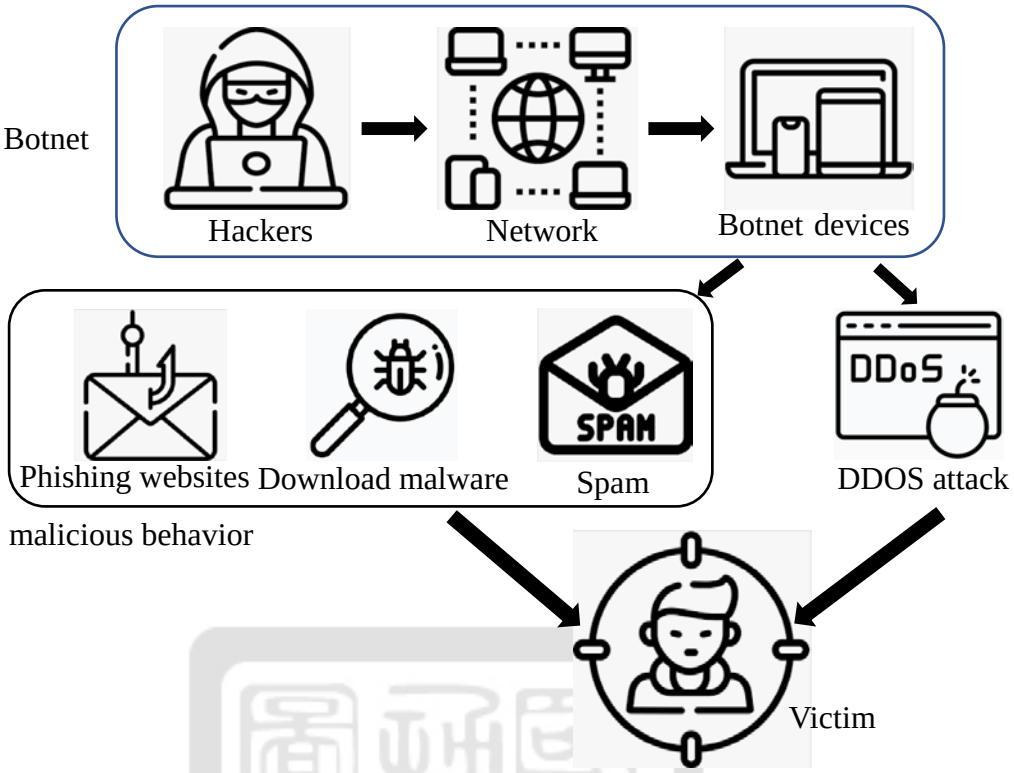


Figure 1-1. Botnet Operation Structure Diagram.

the culprit that caused many websites in the east coast of the United States to crash is the Mirai botnet (Botnet) composed of a large number of IoT devices. In addition, because the scope of its damage is quite large, the scientific and technological editions of major newspapers and magazines have reported and discussed it one after another.

Botnet is the most important threat to information security at present. Its operation structure is shown in Figure 1-1. It consists of a group of network devices that have been implanted with viruses by hackers. The devices that are successfully infected with the virus are called "zombies" (bot), hackers can spread the virus through these zombies to infect more devices. This group of zombies is called "botnet", and hackers can conduct malicious operations through zombies to obtain benefits.

According to a computer security analysis report provided by the American technology company Neustar, the distributed denial of service (DDoS) attacks initiated by botnets will bring deadly information security threats to enterprises and government agencies. DDoS means that the master

controller transmits a large number of legitimate or forged requests to the target network device by controlling the zombie to occupy resources, thus causing the victim to be unable to load and unable to provide normal network services.

In addition, botnets have other attack methods, such as stealing confidential information, kidnapping computers, downloading programs containing viruses, etc. The scale of botnets grows very rapidly and is difficult to detect. It can be seen that the threat posed by botnets cannot be underestimated. A computer infected with a zombie is equivalent to giving hackers control over the computer and allowing hackers All malicious operations performed by the client.

A botnet is formed by hackers controlling a large number of computers infected with zombie viruses, these infected computers are called zombies. Hackers conduct malicious behaviors by manipulating botnets, such as conducting DDoS attacks, stealing personal and confidential information, spreading virus-containing spam, downloading malicious programs, etc., and even renting zombies for DDoS attacks for profit. Botnets will continuously spread virus-containing links or spam emails in social networks, and users will be infected when they click on bot-virus-containing links or spam emails. When a bot virus successfully infects a computer, the computer becomes a member of a botnet, which means that hackers can control the computer to perform malicious actions at will.

A typical botnet is shown in Figure 1-2, consisting of hackers, Command & Control (C&C) servers and zombies. In order to maintain the operation and contact of the botnet, the bot will regularly contact its known C&C server to obtain commands issued by recent hackers or report its own information. The C&C server is the role of the trusted zombie appointed by the hacker. When the hacker wants to perform malicious actions on the target, he only needs to issue the attack command

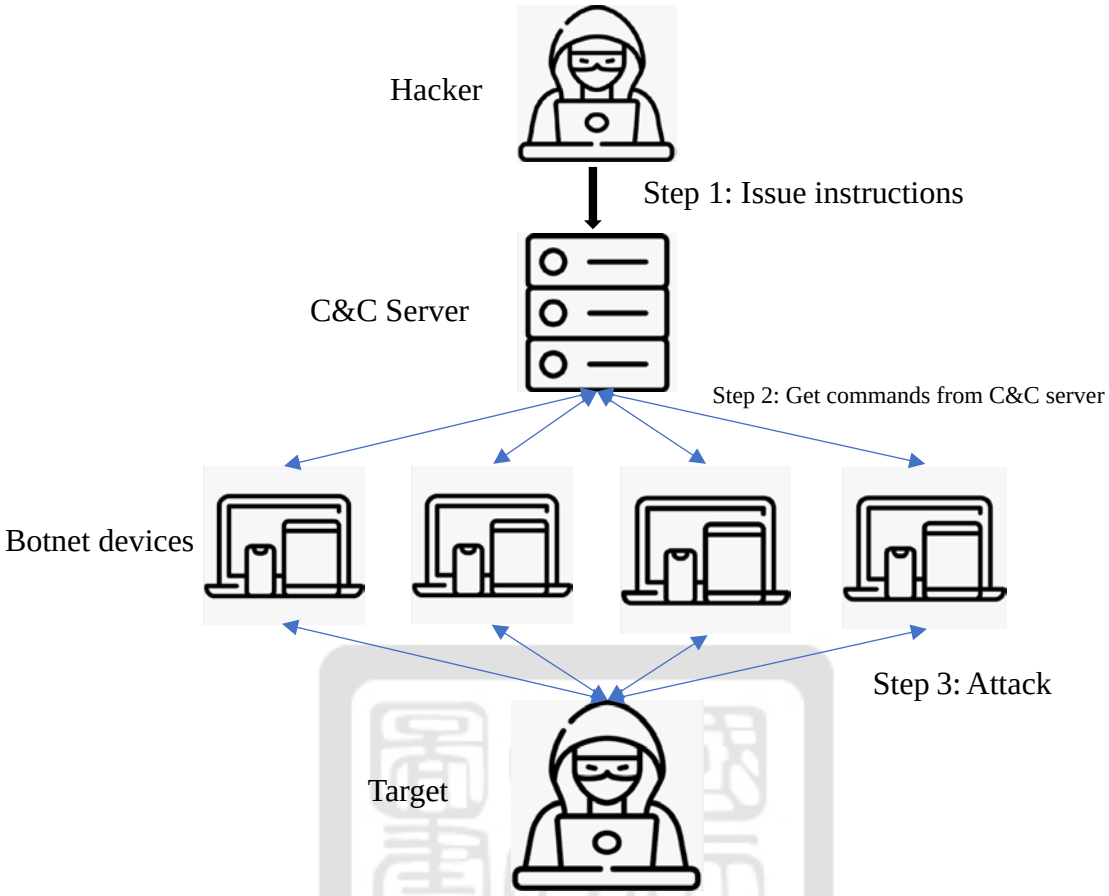


Figure 1-2. Typical Botnet Diagram.

to the zombie who is the C&C server. When you get in touch with a known C&C server, you can get the instructions issued by the latest hacker from it.

This method of transmitting instructions does not require hackers to issue commands to all zombies, which not only improves the speed of transmitting commands, but also only the zombies acting as the C&C server will contact the hackers, thus reducing the communication between all zombies and hackers. contact. In addition, this approach enhances the secrecy of the botnet controller and reduces the chance of exposure.

Threats brought by botnets are mainly related to the characteristics and functions of botnets. The botnets that have evolved so far have certain basic functions and structures. Generally speaking, bots are composed of two modules: main functions and auxiliary functions, as shown in Figure 1-3.

The main function is responsible for the core operation of the bot, such as the network module

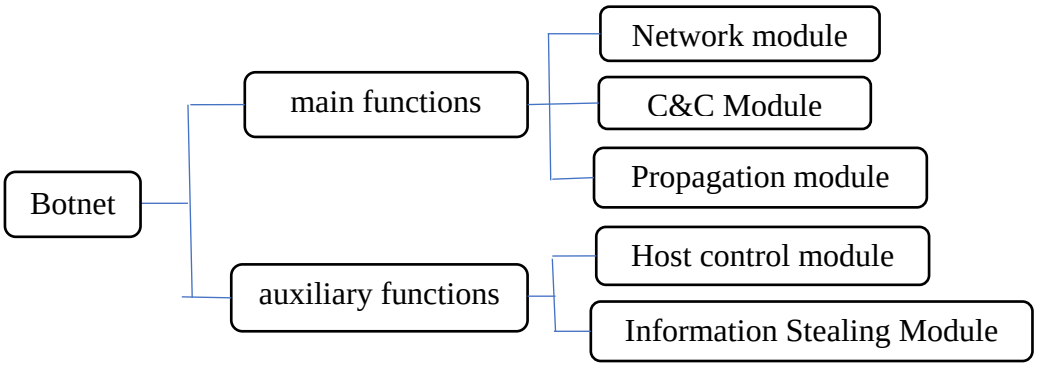


Figure 1-3. Main functions of botnet.

responsible for network communication, which is used to manage the connection of the botnet and the network resources used; the C&C module, which is responsible for processing the information issued by the master. Command, judge the authenticity of the command and whether it is executed; the propagation module is responsible for releasing the zombie virus to infect more computers. The auxiliary function is responsible for the functions outside the core operation of the bot, such as the host control module, including DDoS attack components, proxy server components, web server components, etc.; the information stealing module can steal the personal information stored in the computer; Other functions include anti-honeypot, automatic download and other modules.

Botnets have always been a security threat to the online world. Hackers control zombie computers to launch distributed denial-of-service (DDOS) attacks, causing servers to fail to operate normally, resulting in huge impacts and losses for enterprises or government agencies.

1.2 Objectives

According to the report of Paloalto Network in 2020. Since the beginning of 2020, researchers have noticed that there are 69950 URL related to COVID-19 's epidemic situation. As the epidemic situation of COVID-19 has not stopped, Internet marketing activities continue to favor the news of vaccine development or new restrictions in various countries, and Internet criminals who intend to obtain vaccine information will also continue to use vaccine development companies as targets. In addition, phishing pages designed on the theme of the epidemic, most of these attacks are business certificates for users: for example, Microsoft, Webmail, Outlook, etc., of which Microsoft registration accounts for 23%. Secondly, it has been found that since February 2021, there has been a substantial increase in Google searches and URL related to the COVID-19 epidemic**,** also found that cyberhackers are stealing personal data from these developments.

In the early stage of the pandemic (March 2020), cyber attackers focused on the fields of "COVID-19 epidemic situation Test Kit" and "personal protective equipment (PPE)". For example, due to the shortage of COVID-19 epidemic testing tools in the United States, researchers have found a substantial increase in the number of tools related to detection phishing attacks during this period, many of which are in the form of Internet shopping fraud, such as the built fake Microsoft Sharepoint login network.

As attackers are constantly on the lookout for the latest trends and constantly creating new phishing tactics, cybersecurity defenses should also be adjusted. In addition, hackers will quickly turn newly discovered vulnerabilities into attack weapons, making it difficult for administrators to patch in time. For example, hackers used fake websites from brands like Pfizer and BioNTech to conduct Covid-19-themed phishing attacks that require users to log in with Office 365 credentials to register for the vaccine. This phishing site uses an increasingly common technique known as "Client-Side Cloaking" - instead of immediately popping up a form intended to steal users' personal information, the site first asks the user to

click the "Login" button, to avoid phishing detectors.

If the epidemic does not subside, working from home or distance learning will become the mainstream in the future, and human life will be entirely online. 5G deployment is the best solution to meet the above requirements. However, in the high-speed network environment, it is easier for hackers to jump from one corporate network to other corporate networks, and the home network will also become a springboard for gangsters to attack; hijacking home computers or other devices on the network will eventually hack into the enterprise intranet. This type of supply chain attack will also affect downstream manufacturers and employees who need to remotely access confidential or critical information of the enterprise, thereby allowing sensitive information such as human resources, business, and technical support to be stolen by hackers.

In addition, the attacks of remote hackers on the Internet will turn to routers as the target of attacks. Hacking home routers will become the latest service model for hackers, and then sell access rights to the home network. This kind of "access service" will become the future and will become a business model for hackers to make money. Hackers will take control of the device for a long time and sell the home network access rights of some high-value targets (such as senior executives or system administrators) to criminals; The ultimate target of criminals in the future will be enterprises that have integrated information technology and operational technology networks. Cybercriminals will make profits by selling OT network access rights as their main business.

Chapter 2 Related Work

2.1 Sandbox

The analysis method is to create a virtual environment isolated from the real system, put malicious programs into the virtual environment, and trigger the malicious programs to operate in sandbox. In addition to avoiding the impact of malicious software on the actual system, it can also analyze all behaviors of the malware in detail. The sandbox for dynamic detection and analysis of malicious programs must be safe and reliable, without any impact on the physical system, and the environment can be easily restored to the initial state before the operation of malicious programs, so that other malicious program tests can be carried out smoothly running. In this dynamic analysis mode, the required virtual environment is established in a sandbox.

In terms of technique for dynamic analysis of malicious programs, there are quite a variety of sandboxes for creating environments, and the protection and barrier levels of each sandbox for malicious programs are also different such as limon sandbox, cuckoo sandbox, detux sandbox, joe sandbox and see sandbox. This technique is to determine whether the program is a malicious program by establishing an environment that can execute malicious programs without affecting the operation of the physical system, allowing the program to operate in it, observing the operating behavior and the impact on the environment through the sandbox. In the sandbox, all of the malicious programs are isolated from the physical system or have some control and restrictions. Even if the malicious program runs, it will not affect the physical environment. Therefore, through the complete execution of the malicious program, its behavior and related information can be analyzed as the benchmark information for distinguishing the malicious program.

After the sandbox is established, the researchers will put a potentially malicious program into the virtual environment, trigger the execution of the program, detect whether the program has malicious behavior, and record the function of the program, including the domain name and IP address.

Addresses, file paths, and other characteristics of files related to it on the system or network. In addition, if the malicious program intends to communicate with the attacker/malware developer, it will also record the relevant content of the command to contact the external server, control or attempt to download the rest of the malicious software.

This analysis mode can quickly identify and analyze malicious programs. However, since a virtual environment isolated from the outside world is required, the production and preparation of the relevant environment consumes a lot of money and requires a lot of hardware resources. But relatively speaking, it is possible to obtain the details of the malicious program from generation to result in a clear and detailed manner. Although the amount of data is quite large, the malicious program can be analyzed quickly and correctly with appropriate tools. More efficiently identify and significantly reduce the threat of malicious programs.

Compared with static analysis, dynamic analysis executes the program in an environment that is almost isolated from the physical host and records the behavior of the program. If there is any malicious behavior, it can be clearly judged that the program is malicious. Sandbox can make up for the defects of static analysis. In the dynamic analysis method, a sandbox is mostly established in which malicious programs are executed to obtain more accurate malicious program behaviors. However, since this method needs to execute the malicious program, it takes a long time, the resources and techniques needed to establish a sandbox and operate are higher than static analysis. In addition, if the malicious program includes the function of detecting the virtual environment, it may be detected by the malicious program that it is in the virtual environment, and then it will not execute or perform malicious behaviors to avoid analysis by researchers.

Although the current analysis technology and equipment are becoming more and more complete, as well as the evolution of related tools, most malicious programs can be analyzed smoothly. However, with the improvement of protection and identification tools, malicious programs will also add more protection functions to avoid being analyzed. If only one analysis method is used, it is inevitable that

due to the weakness of the analysis characteristics, there will be misjudgments or situations that cannot be handled. Therefore, the best malware protection effect must be achieved by combining and complementing the two analysis methods. Through the combination of static analysis and dynamic analysis, malicious programs can be effectively and correctly identified and subsequently processed.

2.2 Models

2.2.1 Machine Learning Models

2.2.1.1 Decision Tree [4]

The decision tree uses a greedy rule to decide what questions to ask at each layer. The goal is that after classification, each group can clearly know which category it belongs to. We need objective criteria to decide each branch of the decision tree, so we need to have a judging metric to help us make decisions. The decision tree algorithm can use different indicators to evaluate the quality of branches. Common decision-making disorder evaluation indicators include Information gain, Gain ratio, and Gini index. Our goal is to find a set of decision rules from the training data such that each decision maximizes the information gain. The above indicators are all measuring the degree of confusion in a sequence, and the higher the value, the more chaotic. However, Gini is used by default in the Sklearn suite.

Information Gain maximizes information gain for each decision by finding rules from training data. Its algorithm mainly calculates entropy, so the smaller the amount of information divided by the decision tree, the better. The larger the value of Gini, the more chaotic the data in the sequence is. The value is between 0 and 1, where 0 means that the feature is perfectly classified in the sequence. Entropy is a way to calculate Information Gain. Before understanding Information Gain, it is necessary to understand how entropy is calculated. Entropy is 0 when all the data are classified the same, and Entropy is 1 when half of the data are different. Gini impurity is another measure of chaos, and the higher the number, the more chaotic the data in the sequence. The degree of confusion is 0 when all the data are classified the same, and the degree of confusion is 0.5 when half of the data are different.

2.2.1.2 Logistic Regression [5]

Logistic regression is a variation of linear regression, which is a classification model. The goal

is to find a straight line that can clearly separate and classify all the data, which we can also call a regression linear classifier. Logistic regression is actually explaining the meaning of a probability. A set of parameters obtained by training a function, different w and b will get different functions.

Linear Regression vs Logistic Regression

Logistic regression is used to deal with classification problems. The goal is to find a straight line that can classify data. It mainly uses the sigmoid function to convert the output into a value of 0~1, indicating the probability value that may be this category. Linear regression is used to predict a continuous value, and the goal is to find a straight line that can approximate the real data.

Logistic regression is a basic binary linear classifier. We are looking for a probability (posterior probability) to output predicted Class 1 when the probability $P_{w,b}\ (C_1\,|x)$ is greater than 0.5, and output Class 2 when the probability is less than 0.5. If we assume that the data is a Gaussian probability distribution, we can say that this posterior probability is σ(z). Among them, z=w*x+b, x is the input feature, and w and b are the weight and bias, respectively. They are a set of parameters obtained through training.

The loss function in logistic regression is that the object to be minimized is the sum of the cross entropy of all training data. We want the output of the model to be as close to the target answer as possible. The last step is to find the best set of parameters so that the loss can be the lowest. Therefore, Gradient Descent is used here to minimize Cross Entropy.

2.2.1.3 Random Forest [6]

Suppose we have a thousand pieces of data, and we want to extract 100 pieces of data from it, and there may be duplicate data in these 100 pieces of data. Then the second step selects k features from these extracted data as the post-selection of decision factors, so each tree can only see part of the features. The third step repeats the above steps m times and generates m decision trees. Repeating the Bootstrap step m times, we will have m sets of training data, and each set of training data has n' pieces of data. Finally, through the decision of each tree and a majority vote, the final predicted

category is determined. Because the number of features of each tree in the random forest may be different, the final decision results will be different. Finally, according to the different tasks, we will do regression or classification problems. If it is a regression problem, we will average the output of these decision numbers to get the final answer. If it is a classification problem, we will use a majority vote to integrate all tree predictions. the result of.

The advantages of random forest include that

1) It is based on the CART algorithm, so it can process both categorical and continuous data.
2) For most of the data, the accuracy of the fitting results of the random forest algorithm is high.
3) Accept high-dimensional feature data.
4) Using the Bagging sampling method to perform error analysis in the OOB method can improve the computing efficiency.
5) After training, it can draw feature importance.

2.2.1.4 Gaussian Naïve Bayes [7]

Gaussian Naive Bayes (GNB) is a classification technique for machine learning based on probabilistic methods and Gaussian distributions. Gaussian Naive Bayes assumes that each parameter has an independent ability to predict the output variable. The combination of predictions for all parameters is the final prediction, which returns the probability of the dependent variable being classified into each group, with the final classification assigned to the grouping (class) with higher probability.

The so-called Gaussian Bayesian refers to assuming that the conditional probability of each feature dimension of the sample obeys the Gaussian distribution, and then calculates the posterior probability of the new sample belonging to each category under a certain feature distribution according to the Bayesian formula. Finally, the category of the sample is determined by maximizing the posterior probability.

The advantages of Gaussian Naïve Bayes include that

1) The algorithm logic is simple and easy to implement.

2) The time and space overhead in the classification process is small.

However, Gaussian Naïve Bayes also has disadvantages. The Naive Bayesian model assumes that the attributes are independent of each other. This assumption is often not true in practical applications. When the number of attributes is large or the correlation between attributes is large, the classification effect is not good.

2.2.1.5 Support Vector Machine (SVM) [8]

SVM is an algorithm proposed by a famous binary classifier based on statistical learning theory. To put it simply, SVM is a supervised learning algorithm that tries to construct a decision hyperplane between two categories, divide the data into two categories (2 classes), and finally classify or predict, so it is also known as a binary classifier. Mathematically speaking, finding a decision boundary maximizes the margin of the two categories. The wider the margin, the better the model will be for new data. The sample points that fall on the boundary and within the boundary are called is the support vector. If the hyperplane can perfectly cut the data, it is called linear separability; otherwise, it is called non-linear separability. In practice, the latter is more common.

In two-dimensional space, a hyperplane refers to a straight line; in three-dimensional space, it refers to a plane. The reason why there is a word "super" is because the actual data is often two-dimensional or three-dimensional. In higher dimensional space, we cannot observe what the shape of this plane is, so the term "hyperplane" is used to summarize all situations.

Whether it is the application of small samples (belonging to the category of statistical learning), nonlinear separability, high-dimensional pattern recognition problems, SVM has a very good performance.

2.2.1.6 K-Nearest Neighbor (KNN) [9]

Its idea is that when the data and labels are input into the test data in the known training set, compare the features of the test data with those of the related training data set, and find the top K

most similar data in the training set, then the category related to the test data is the event that occurs in most cases in the k data.

Since the KNN model mainly relies on limited peripheral samples rather than the method of discriminating the class domain to determine the category to which it belongs, the KNN model is more accurate than other methods for the sample sets to be divided when the class domain crosses or overlaps more. The KNN algorithm can be used not only in classification, but also in regression. Trying this algorithm is a good way to benchmark before considering more advanced techniques. Building a k-nearest neighbor model is usually fast, but if the training set is large (with a large number of features or samples), the prediction speed may be slow. When using the KNN algorithm, it is very important to preprocess the data. This algorithm often does not work well for data sets with many features (hundreds or more), and most values for most features are 0. This algorithm works especially poorly for small datasets (so-called sparse datasets). By finding the k nearest neighbors of a sample and assigning the average value of the attributes of these neighbors to the sample, the attributes of the sample can be obtained. A more useful method is to give different weights to the influence of neighbors with different distances on the sample, such as the weight is inversely proportional to the distance.

The advantages of KNN include

(1) Retraining is cheap (no model building)

Just save training samples and labels, no parameter estimation, no training required.

(2) It is suitable for automatic classification of large samples.

Although the k-nearest neighbor algorithm can solve multi-classification problems very well, it also has many shortcomings, mainly in the following aspects:

(1) Inefficiency:

For each prediction data, the time complexity of O(m*n) is required, and it can be optimized using the tree structure, but even after optimization, its efficiency is relatively low.

2.2.2 Deep Learning Model

Convolutional Neural Network (CNN) [10]

Artificial Neural Network (ANN or Neural Network, NN) is a mathematical model or computational model that imitates the structure and function of a biological neural network. Due to the gradual maturity of the development of computer science, the new era of artificial intelligence is coming. The main method is to use computer programs to simulate the behavior of neurons transmitting signals in large quantities, and to input all the information and calculate and correct it repeatedly. However, the rigid and large amount of calculations, even if the hardware structure and calculation methods used by the latest computer technology are used, still takes too much time. Neural network-like algorithms, which imitate the operation of the human brain to allow computers to perform calculations, use the concept of multi-nodes and layers to extract data features for modeling.

So a new neural network called Convolutional Neural Network (CNN) is proposed for pictures or other multi-dimensional data types. CNN can be applied to image, sound and other signal types of data types to obtain good results. CNN is one of the most common deep learning architectures, because the convolutional layer and pooling layer strengthens pattern recognition and the relationship between adjacent data.

The design concept of convolutional layer is to judge the adjacent data according to the characteristics of the data; taking the two-dimensional data type as an example, the adjacent data does not only have the front and rear directions like text, but has the plane direction of up, down, left, and right. To analyze the picture block, instead of dividing the picture into pixel-by-pixel calculations, it is to think in units of blocks to find out the expected features in the picture. The pooling layer is to reduce the dimensionality of the signal. After mathematical operations, the higher the value, the more similar it is to the feature. In other words, what kind of feature appears in what position of the picture, the higher the value, that is, the closer the value is to 1, the more similar the area is to the target feature. Although the next step can be directly judged after obtaining the location information, in fact,

so much location information is not needed, so a structure called a pooling layer is derived. The pooling layer simplifies the numerical partitions, that is, the regions that are not similar to the features are merged and omitted, and at the same time, the approximate location of the target features is simplified and outlined. In this way, a large number of unnecessary calculations are reduced, and the convolution layer is repeatedly executed to remove the chaff from the picture, and then use the traditional neural network to perform the final judgment.

2.2.3 Model Performance

The performance of the classifiers is evaluated by Accuracy, Area under the ROC curve, Confusion Matrix, F1-Score, Precision and Recall. The macro-averaged precision is computed using the arithmetic mean of all the per-class precisions. The macro-averaged recall is computed using the arithmetic mean of all the per-class recalls. The macro-averaged F1 score (or macro F1 score) is computed using the arithmetic mean of all the per-class F1 scores. This method treats all classes equally regardless of their number of testing ELF files values. The weighted-average precision is calculated by taking the mean of all per-class precisions while considering each class's number of testing ELF files values. The weighted-average F1 score is calculated by taking the mean of all per-class recalls while considering each class's number of testing ELF files values. The weighted-average F1 score is calculated by taking the mean of all per-class F1 scores while considering each class's number of testing ELF files values. We define the following measures:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

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

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

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

$$True\ Positive\ Rate\ (TPR) = \frac{TP}{TP + FN}$$

$$True\ Negative\ Rate\ (TNR) = \frac{TN}{TN + FP}$$

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{TN + FP} = 1 - TNR$$

$$False\ Negative\ Rate\ (FNR) = \frac{FN}{TP + FN}$$

where

True Positive (TP): The tested case is classified correctly as benign, and it is the same as the target we observed. In our experiments, TP means that the benign samples are truly predicted to be benign.

True Negative (TN): The tested case is classified correctly as malware, but it is the same as the target we observed. In our experiments, TN means that the malware samples are truly predicted to be malwares.

False Positive (FP): The tested case is classified incorrectly, but it is not the same as the target we observed. In our experiments, FP means that the malware samples are incorrectly predicted to be benign.

False Negative (FN): The tested case is classified incorrectly to malware, and it is not the same as the target we observed. In our experiments, FN means that the benign samples are incorrectly predicted to be malwares.

The confusion matrix is an N*N matrix (N = number of target classes) in which $num_{i,j}$ of i^{th} row and j^{th} column indicates there are $num_{i,j}$ tested samples that are predicted to be class *j* but they actually class *i*, assuming x-axis corresponds to actual class and y-axis corresponds to predicted class. In Figure 2-1 shows that our experiment contains two classes, benign and malicious, so the confusion matrix presents 2*2 confusion matrix. It helps to evaluate the performance of classification models.

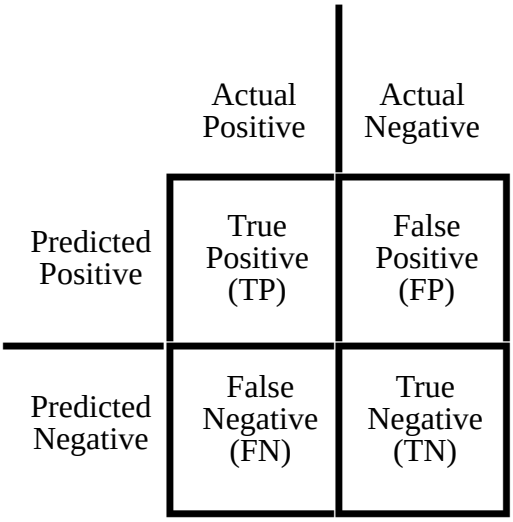


Figure 2-1. Confusion matrix.

The so-called "good model" is a model with a high probability that the actual label is consistent with the predicted label.

ROC (Receiver operator characteristic) is also known as "receiver operator characteristic map", its definition is a graph-based analysis tool for selecting the best signal detection model, discarding the next-best model, or setting the best threshold within the same model. AUC (area under the curve) indicates that, when a positive sample is randomly selected, the probability that the classifier will correctly judge it as a positive sample is higher than the probability of judging it as a negative sample. So, the higher the AUC, the higher the correct rate of the classifier. Generally, when judging the quality of the test tool, in addition to looking at the graph of the curve, we can also use the AUC to judge the discrimination of the ROC curve. The value of AUC ranges from 0 to 1, and the value is the bigger the better. When AUC=0.5, it means no discrimination. When $0.7 \leqq AUC \leqq 0.8$, it means acceptable discrimination. When $0.8 \leqq AUC \leqq 0.9$, it means excellent discrimination. When $0.9 \leqq AUC \leqq 1.0$, it means outstanding discrimination.

Chapter 3 Environment

3.1 Differences between Sandbox and Virtual Machine

Virtual machines share many of the same characteristics as sandboxes, making them easy to confuse. Before discussing the difference between a virtual machine and a sandbox, let's first look at the general structure of a general computer. Figure 3-1 shows the operating system windows serve as a bridge between the application program and the driver of the hardware. The application runs on Windows and interacts with the user and the machine hardware through Windows (before the operating system appeared, only professionals could operate computers, and it is precisely because of the emergence of operating systems that ordinary people can also operate computers). In addition, the operating system is responsible for accessing file resources on the hard disk and controlling the machine hardware through drivers. There are many drivers, and different drivers need to be configured for different hardware. Figure 3-2 shows computer structure with sandbox. The sandbox is characterized by system isolation. To some extent, the sandbox can be regarded as a container, the application runs in the sandbox, and the sandbox runs on the windows operating system. Applications running in the sandbox can access resources such as files in the hard disk just like applications outside the sandbox. The main difference between the application running in the sandbox and the application

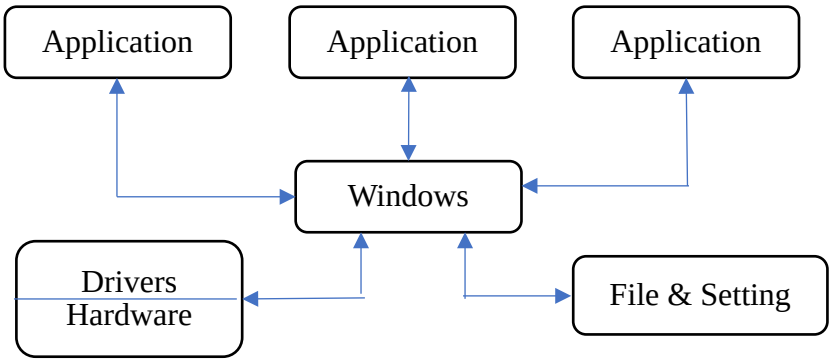


Figure 3-1. General computer architecture.

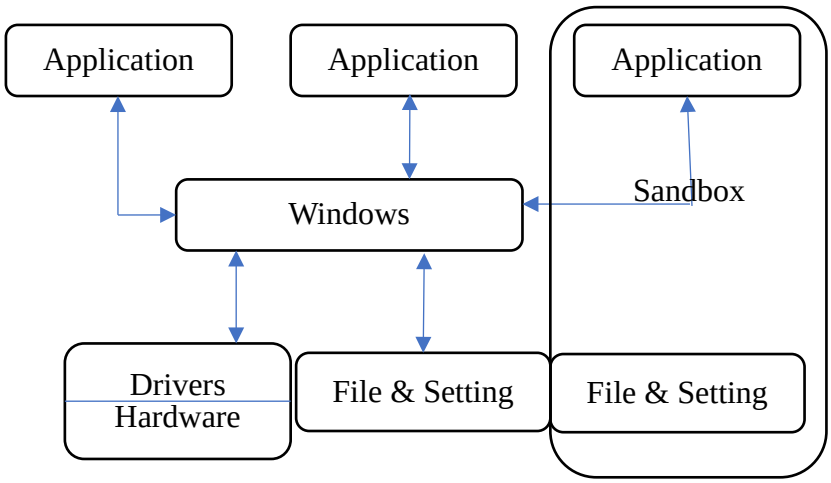


Figure 3-2. Computer structure with sandbox.

outside the sandbox is that for the application outside the sandbox, the application inside the sandbox is invisible, and when the application inside the sandbox exits, changes made will not be saved.

The virtual machine is referred to as VM, which is essentially an application running on the operating system. Its special feature is that the virtual machine simulates a complete and independent computer environment (but not a real computer environment) through software methods. This is also the origin of the name of the virtual machine. Therefore, the virtual machine is like a copy of the real computer, and the virtual machine can be regarded as a "machine inside the machine". For example, we often install a Linux virtual machine on the windows operating system. In this case, Windows is called a host, and Linux is called a guest. Multiple guest virtual machines can be installed on a host, just like windows can be installed with multiple applications. Figure 3-3 shows Computer Architecture with virtual machine. The application running in the sandbox and the application outside the sandbox share the storage memory and computing resources of the machine.

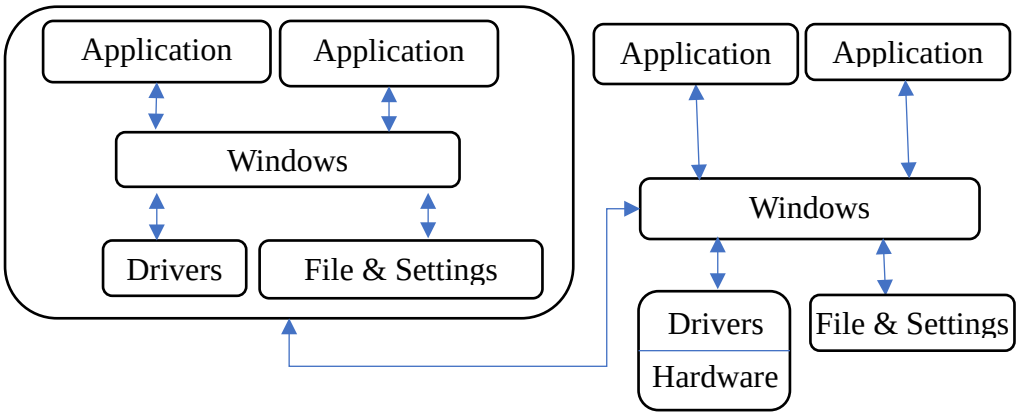


Figure 3-3. Computer Architecture with virtual machine.

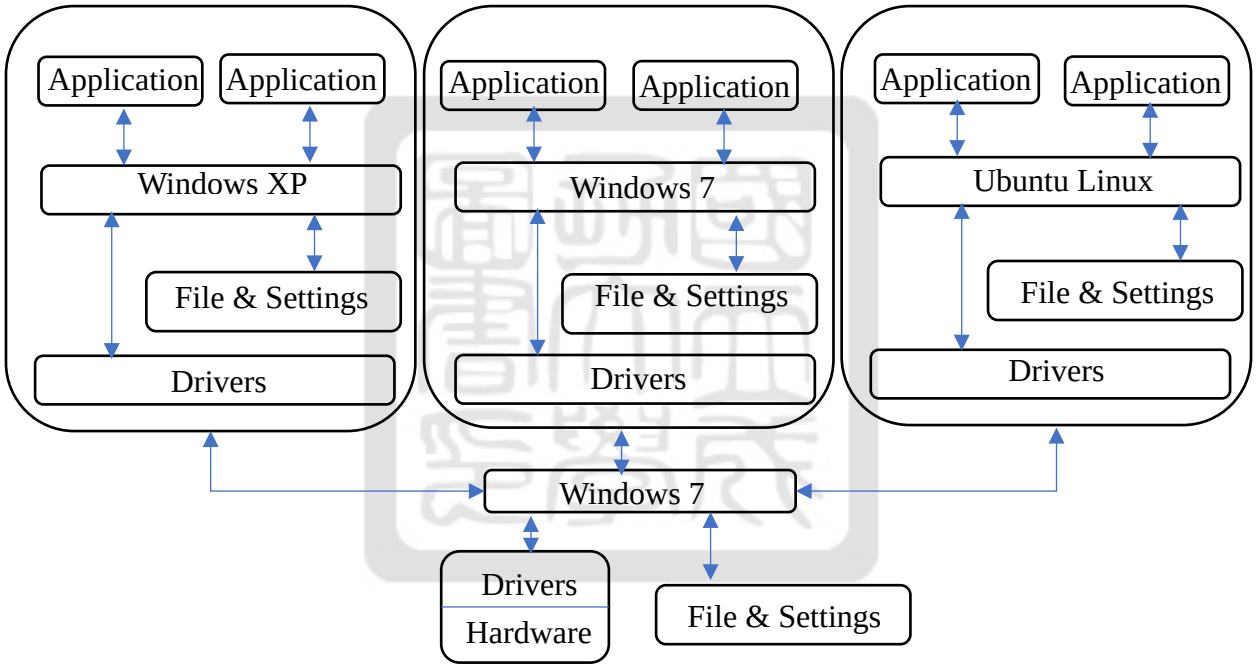


Figure 3-4. Computer structure with multiple virtual machines.

Figure 3-4 shows Computer structure with multiple virtual machines. In a virtual machine, many applications can be installed, and these applications can use resources inside the virtual machine, including virtual hard disks and computing resources. In addition, the virtual machine can have its own driver, so the virtual machine is used as an application, using real hardware resources to simulate its own hardware resources and software environment. Therefore, when the virtual machine exits, the changes (including downloads, settings, updates, installations, etc.) that occurred inside the virtual machine will be saved.

We can see that the main differences between virtual machines and sandboxes are:

1. When the application in the sandbox exits, its changes will be discarded; and when the virtual machine exits, its changes will be saved.

2. The application in the sandbox shares the hardware resources of the machine with other applications; while the virtual machine needs to specify the memory and CPU core for it during installation, and the virtual machine does not share hardware resources with other applications. Therefore, the virtual machine actually consumes a lot of system resources.

The advantages of the sandbox include the following:

(a) Security: The operation of the virtual environment in the sandbox is exactly the same as that of the local environment, but the sandbox and the local environment are completely separated in operation and do not affect each other, as if two computers exist on one computer at the same time generally. This feature is often used by forensics or information security personnel to open suspicious applications or unsafe websites, because it will not affect the local environment. Even if the sandbox environment is accidentally poisoned, it only needs to be closed.

(b) One-time: When the sandbox is closed, any operations during the period of opening the sandbox, such as adding files, folders, or searched webpage records, will be cleared at the same time, and it is not easy to use other Ways to retrieve the deleted data content. When the sandbox is opened next time, it will be a brand-new sandbox environment.

(c) Saving time and cost: Before the sandbox function is released, if you want to use the virtual environment, you must install additional related software and use image files to install the operating system you want to install. However, after the launch of the sandbox, you only need to enable the function of the sandbox to achieve the effect of being like a virtual machine, which saves a lot of time and cost compared with traditional virtual machines.

(d) Installation complexity: To use the sandbox, you only need to enter the computer BIOS to virtualize the core of the operating system. But if you want to install a traditional virtual machine,

you must first install the virtual machine platform, obtain the installed CD image file, and initialize the installed operating system after the installation. Therefore, installing a sandbox is relatively easy and low complexity.

(e) Space occupied by the hard disk: Whether it is the platform or the virtual machine itself, the virtual machine will occupy a large amount of space in the computer (because it is mostly a complete system), and when the space in the computer is occupied too much, it may cause The boot speed becomes slow or the file you want to save cannot be saved. The sandbox is less prone to this situation, because when the sandbox is not enabled, it only occupies about 25MB of system space, which is very lightweight compared to the virtual machine.

(f) Memory Occupancy: Both the traditional virtual machine and the sandbox will occupy the memory of the local computer when running. However, when too many applications are opened, the traditional virtual machine will still occupy the memory of the local computer. The sandbox can flexibly relinquish the memory space to the local computer, so that the local computer will not slow down due to the sandbox when running other applications.

In addition, the sandbox also has the following disadvantages:

(g) Number of open sandboxes: Sandboxes can only open one sandbox on a computer at the same time, unlike traditional virtual machines, as long as the computer has enough hardware space, multiple devices can be opened at the same time. Therefore, when multiple virtual environments must be established simultaneously, the requirements cannot be met.

(h) Additional configuration files must be created: the sandbox default supports simple configuration files, allowing users to use the most basic functions, but if you want to use some more advanced functions, such as using GPU, setting network access, etc., you must It is more troublesome and more difficult to create additional configuration files.

Table 3-1. Analysis tool for Linux elf files.

tools	function
strace	Tracing system calls generated and received signals during program execution
readelf	Used to view file information in ELF format
file	Used to identify file types
objdump	Gives more information in a human-readable format about additional information the binary may have
ldd	List the dynamic link libraries required by a program
hexdump	hexdump is mainly used to view the hexadecimal encoding of "binary" files
ar	Create static library, insert/delete/list/extract member functions
strings	List all printable strings in object file
nm	List the symbols defined by the symbol table in the object file
strip	Remove symbol table information from object file
size	List the size of each segment in the object file

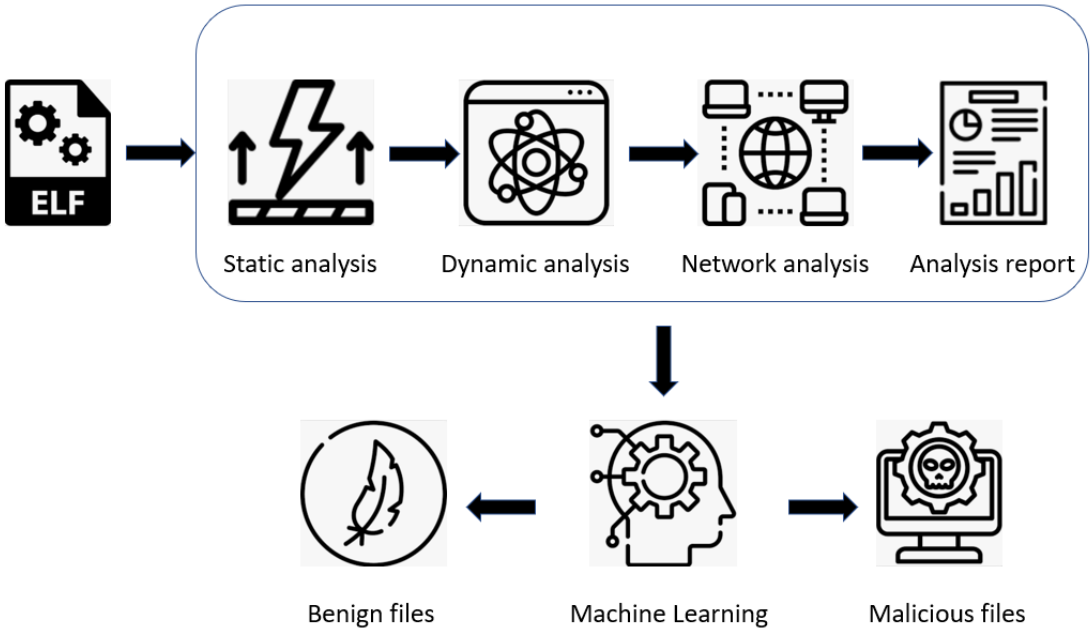


Figure 3-5. The framework of the proposed scheme.

3.2 Environment for experiment

To perform our experiments, we used **Limon sandbox** for analyzing Linux ELF files. **Limon sandbox** is a research project writing a development sandbox in python that automatically collects, analyzes and reports runtime indicators of Linux files. Limon sandbox performs static analysis, dynamic analysis, and memory analysis by using open-source tools, such as *yara-python*, *ssdeep*, *readelf*, *inetsim*, *tcpdump*, *strace* and *sysdig*. Linux files can be checked before, during and after execution.

The strace command is a tool that integrates diagnosis, debugging, and statistics. It is often used to track system calls and received signals during process execution. We can use it to monitor the interaction between user space processes and the kernel. Such as tracking and analyzing the system calls, signal transmission and process state changes of the application program, so as to solve the problem. Strace is often used to trace system calls and received signals during process execution. In Linux, a process cannot directly access hardware devices. When a process needs to access hardware devices (for example, read disk files, receive network data, etc.), it must switch from user mode to kernel mode, and access hardware devices through system calls. Strace can trace the system calls generated by a process, including parameters, return values, and execution time.

Limon sandbox analyzes malware in a controlled environment by monitoring its activity and child processes to determine the nature and purpose of the malware. It determines the malware's process activity, interaction with the file system, network, and also performs memory analysis and stores the analyzed artifacts for later analysis. Limon involves conf.py, dyan.py, limon.py, meman.py and statan.py.

The program conf.py is responsible for configuring the path of the host machine and analysis machine, where *report_dir* is the path to the directory in which the final report is stored, *virustotal_key* contains the virustotal public api key, *host_analysis_vmpath* contains the path of the .vmx file of the

analysis machine, *host_vmrunpath* contains the path of the vmrun of the host, *analysis_username* is the username set when setting up the analysis machine, *analysis_password* is the password set when setting up the analysis machine, *analysis_clean_snapname* is the cleanup snapshot name set when setting up the analysis machine, and so on.

The program statan.py is responsible for static analysis, recording file type, file size, md5, elf header, program header, section header, symbols, and all these functions are in class Static.

The program dyan.py is responsible for the monitor network part, recording the network behavior of Linux files during execution, including class Inetsim, class Vmware, class Tshark, class Tcpdump, class Fileregmon and class Iptables.

The program meman.py is responsible for extracting digital artifacts from volatile memory (RAM) samples. Extraction techniques are performed completely independent of the system under investigation, yet provide visibility into the runtime state of the system, including class Volatility.

Chapter 4 Experiments and Results

4.1 Dataset

We have collected four types of ELF files in the thesis. There are 1551 benign files consisting of 1180 and 371 system files in directories of /bin and /sbin of Ubuntu 20.04 operating system) and malicious files consisting of 2778 VirusTotal files (i.e., IoT Botnet) from online virus repositories [3] and 475 Linux Malware files [4]. The number of the samples for each type are shown in Table 4-

Table 4-1. ELF collection.

Class	Files	Quantity
Malicious	IoT Botnet Files Linux Malware Files	2778 VirusTotal 475
Benign	System Files	1180 ubuntu bin files 371 ubuntu sbin files
Total		4804
Problematic files	IoT Botnet Files	46
Used in method 1		4758

1.

The /bin directory contains binaries for use by all users. Binaries needed for normal/standard system functioning at any run level. The ‘/bin’ directory also contains executable files, Linux commands that are used in single user mode, and is a place for most common used terminal commands that are used by all the users, like cat, cp, cd, ls, mount, rm, etc. The /sbin contains binaries to configure the operating system. Binaries needed for booting, low-level system repair, or maintenance (run level 1 or S). The ‘/sbin’ directory also contains executable files, but unlike ‘/bin’ it only contains system binaries which contains important administrative commands that should generally only be employed by the superuser to perform certain tasks and are helpful for system maintenance purpose. e.g. fsck, root, init, ifconfig, etc. Many of the system binaries require root privilege to perform certain tasks.

Table 4-2. Linux elf files memory management.

name	significance
.text	The machine code of the compiled program.
.rodata	Read-only data.
.data	Initialized global and static variables.
.bss	Uninitialized global and static variables.
.symtab	A symbol table that stores information about functions and global variables defined and referenced in the program.
.rel.text	A list of locations in the .text section that need to be modified when the linker combines other files with the object file.
.rel.data	Relocation information for all global variables referenced or defined by the module.
.debug	A debug symbol table whose entries are local variables and type definitions defined in the program.
.line	Mapping of line numbers in the original C source program and machine instructions in the .text section.
.strtab	A string table.

We use the feature set to train Machine Learning Algorithms includes Decision Tree, Logistic Regression, Random Forest, Gaussian Naïve Bayes, SVM and KNN for our experiments. We transform our dataset to images then we use CNN to analyze them.

We have three kinds of methods to analyze ELF files include the ELF files has been analyzed by the sandbox, the ELF files analyzed statically, and the ELF files transform to gray scale images directly. We will show each scenario of the classification metrics.

4.2 Experiments using sandbox

We used 70% of ELF files (4758*0.7 = 3330) for training and 30% of ELF files (4758 – 3330 = 1428) for testing. We individually examine the results from Decision Tree, Logistic regression, Random Forest, Gaussian Naïve Bayes, Support Vector Machine, K-nearest neighbor based on the static features, system call features, and combined features including both static and system call features. For all benign and malicious files, the accuracy is the percentage of the accurate predictions.

Table 4-3. Table of feature set.

Feature Type	List of Features
Static Features	Number of Program Headers, Number of section headers, Section header string table index, File Size, Size of .text Section, Size of .data Section, Size of .rodata Section, Size of .bss Section
System Call Features	execve, read, close, write, connect, chdir, bind, unlink, dup2, dup, socket, kill, rename, pipe, accept, creat, fork, open, clone

Table 4-4. System call features for analyzed by sandbox.

System call	description
execve	run another specified program
read	reads data from an open device or file
close	close the file descriptor and release its corresponding kernel resources
write	writes data to an open device or file
connect	connect to remote host
chdir	change current working directory
bind	bind socket to port
unlink	delete link
dup2	copy the file description word according to the specified conditions
dup	copy open file descriptors
socket	create socket
kill	send a signal to a process or process group
rename	file rename
pipe	create pipeline
accept	respond to socket connection request
creat	create new file
fork	create a new process
open	open a file
clone	create child processes according to specified conditions

	file	benign malicious	Number of program headers	Number of section headers	Section
0	aa-enabled	0	13	30	
1	aa-exec	0	13	30	
2	aconnect	0	13	31	
3	acpi_listen	0	13	30	
4	addpart	0	13	30	
...	...	...	...	...	
4753	VirusShare_ff4bdef83f191cd6451e26a09311bcd2	1	6	30	
4754	VirusShare_ff4dbe26278bfda759ee8b1f10d94d3b	1	6	30	
4755	VirusShare_ff8dd015bb2766352af051944cb9781f	1	5	22	
4756	VirusShare_ffb00447d40b0ae015752dd484d09de8	1	6	34	
4757	VirusShare_ffc7be26912b5aca63e55dc7c830f28a	1	7	33	

4758 rows × 10 columns

Figure 4-1. Static features dataset analyzed by sandbox.

	file	benign malicious	execve	read	close	write	connect	chdir
0	aa-enabled	0	1	5	6	1	0	0
1	aa-exec	0	1	2	3	1	0	0
2	aconnect	0	1	58	57	1	2	0
3	acpi_listen	0	1	2	2	0	1	0
4	addpart	0	1	3	4	4	0	0
...	...	...	...	...	...	...	...	...
4753	VirusShare_ff4bdef83f191cd6451e26a09311bcd2	1	1	1	3	2	0	1
4754	VirusShare_ff4dbe26278bfda759ee8b1f10d94d3b	1	1	1	2	2	0	0
4755	VirusShare_ff8dd015bb2766352af051944cb9781f	1	1	0	0	0	0	0
4756	VirusShare_ffb00447d40b0ae015752dd484d09de8	1	1	1	2	1	0	0
4757	VirusShare_ffc7be26912b5aca63e55dc7c830f28a	1	1	12	15	5	5	0

4758 rows × 21 columns

Figure 4-2. System call features dataset analyzed by sandbox.

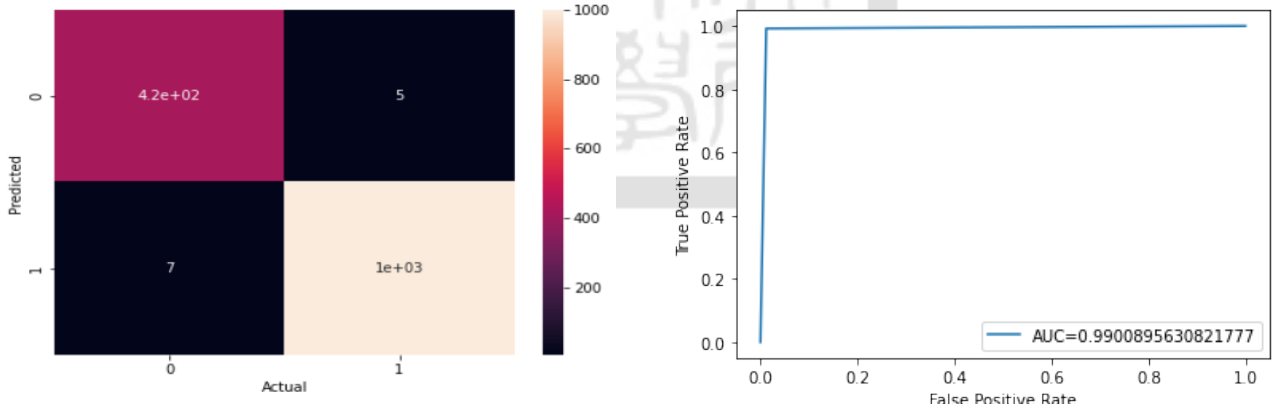
	file	benign malicious	Number of program headers	Number of section headers	Section header string table index	File Size	Size of .text Section	Size of .data Section	Size of .rodata Section	Size of .bss Section	...	dup	socket	kill
0	aa-enabled	0	13	30	29	31248	10389	240	809	32	...	0	0	0
1	aa-exec	0	13	30	29	35344	11717	240	1321	88	...	0	0	0
2	aconnect	0	13	31	30	22912	4757	16	1356	48	...	0	0	0
3	acpi_listen	0	13	30	29	19016	2789	296	647	80	...	0	0	0
4	addpart	0	13	30	29	30952	8773	20	576	48	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4753	VirusShare_ff4bdef83f191cd6451e26a09311bcd2	1	6	30	27	20340	3184	160	352	32	...	0	1	0
4754	VirusShare_ff4dbe26278bfda759ee8b1f10d94d3b	1	6	30	27	20084	2060	48	423	32	...	0	0	0
4755	VirusShare_ff8dd015bb2766352af051944cb9781f	1	5	22	21	11652	5580	76	1531	4912	...	0	0	0
4756	VirusShare_ffb00447d40b0ae015752dd484d09de8	1	6	34	31	19823	3780	12	2536	28	...	0	0	0
4757	VirusShare_ffc7be26912b5aca63e55dc7c830f28a	1	7	33	30	34913	13200	256	4275	1680	...	0	1	0

4758 rows × 29 columns

Figure 4-3. Combined features dataset analyzed by sandbox.

4.2.1 Decision Tree

We examine the results of decision tree model with gini criterion trained individually on the static, system call, and combined features. For static features, our results using gini criterion is shown in Figure 4-4 and Table 4-5. In Figure 4-4(b), we can see that AUC=0.990. In Table 4-5, Benign-precision=416/(416+7)=0.983, Benign-recall=416/(416+5)=0.988, Malicious-precision=1000/(5+1000)=0.995, Malicious-recall=1000/(7+1000)=0.993, macroavg-precision=(0.98+1.00)/2=0.99, macroavg-recall=(0.99+0.99)/2=0.99, weightedavg-precision=(0.98*421+1.00*1007)/1428=0.99, weightedavg-recall=(0.99*421+0.99*1007)/1428=0.99, Benign-f1score=2*0.983*0.988/(0.983+0.988)=1.942408/1.971=0.985, Malicious-f1score=2*0.995*0.993/(0.995+0.993)=1.97607/1.988=0.994. The detailed decision trees built by our programs is also shown in Figure 4-6. As we can see in Table 4-5 that we get accuracy of **0.99** for gini criterion.



(a). (b).

Figure 4-4. Decision tree with **gini** criterion analyzed by sandbox for static features. (a) confusion matrix (b) ROC AUC graph.

Table 4-5. Decision tree model performance with **gini** criterion for static features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.98	0.99	0.99	0.988	421
Malicious	1.00	0.99	0.99	0.993	1007
macro avg	0.99	0.99	0.99	**0.99**	1428
weighted avg	0.99	0.99	0.99	**0.99**	1428

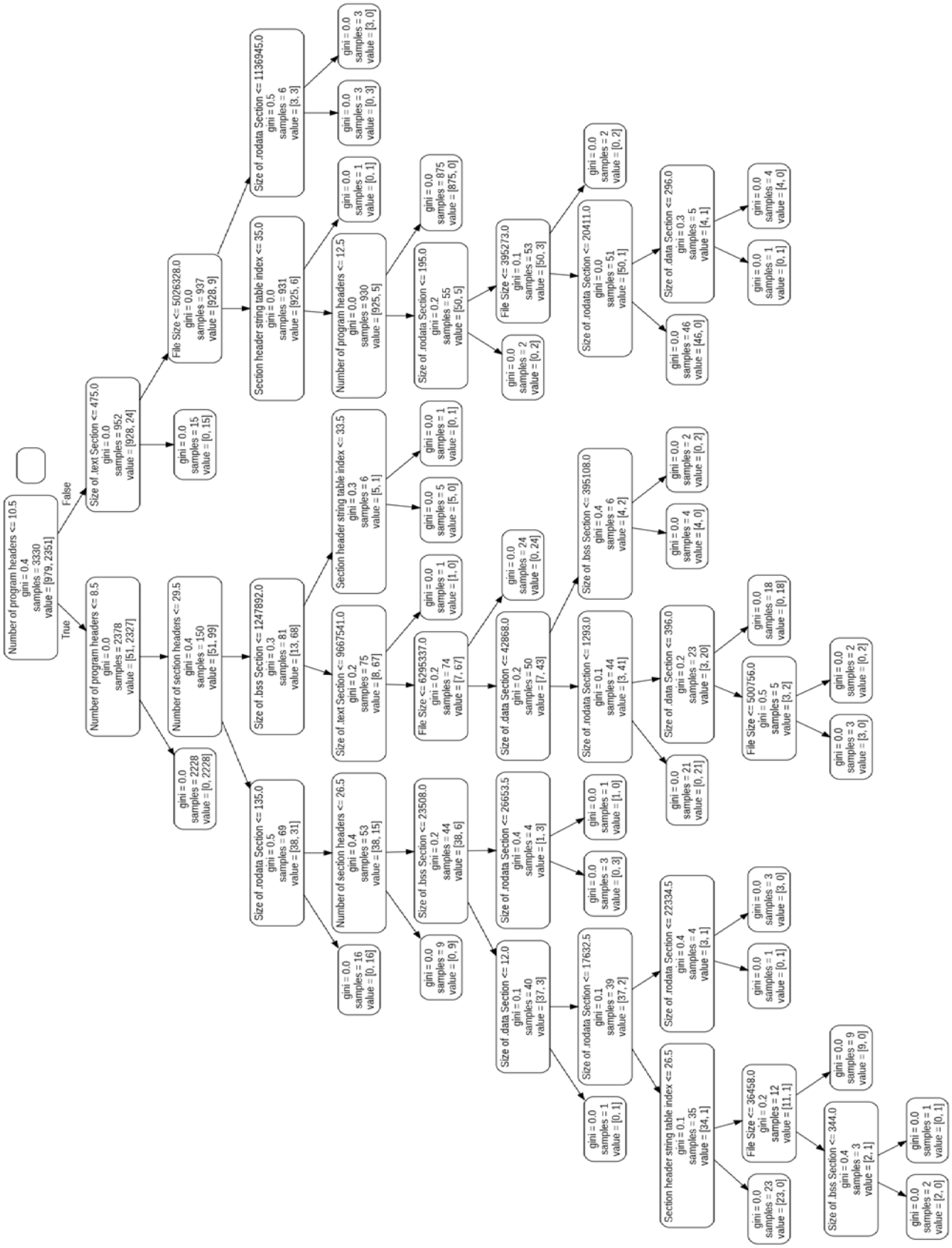


Figure 4-5. Decision tree with **gini** criterion analyzed by sandbox for static features plot tree.

For system call features, our results using gini criterion is shown in Figure 4-6, and Table 4-7. In Figure 4-6(b), we can see that AUC=0.934. In Table 4-7,

Benign-precision=366/(366+51)=0.878, Benign-recall=366/(366+55)=0.869,

Malicious-precision=956/(55+956)=0.946, Malicious-recall=956/(51+956)=0.949,

macroavg-precision=(0.88+0.95)/2=0.91, macroavg-recall=(0.87+0.95)/2=0.91,

weightedavg-precision=(0.88*421+0.95*1007)/1428=0.93,

weightedavg-recall=(0.87*421+0.95*1007)/1428=0.93,

Benign-f1score=2*0.878*0.869/(0.878+0.869)=1.525964/1.747=0.873,

Malicious-f1score=2*0.946*0.949/(0.946+0.949)=1.795508/1.895=0.947. The detailed decision trees built by our programs is also shown in Figure 4-7. From Figure 4-7, it can be found that there is still no way to completely distinguish between benign files and malicious files in the value = [benign, malicious] in the red box of the leaf node, so the accuracy can only reach **0.93**. Take value= [1, 5] as an example. There is one benign file, hciconfig, that cannot be distinguished. And there are five malicious files that cannot be distinguished as follows.

ef30be17cdc69e67d2c82ac2e002190698e337595bf42cbe65e10f5794fa6de2,

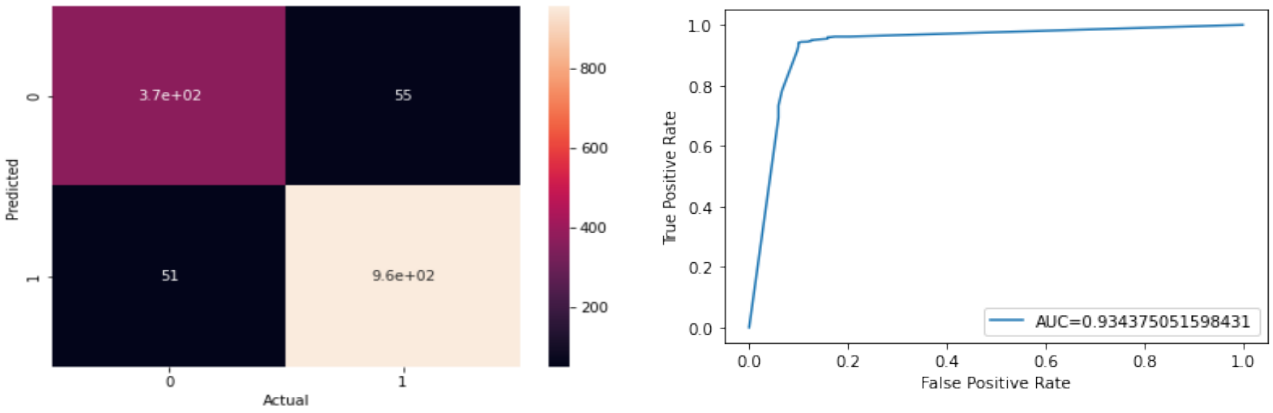
ELF_8F83A5899F8D7C93BEA65F8BC7BCF25C, VirusShare_391c74e627dfea7d3a5e1e9ba807a2d8,

VirusShare_a3f194088d346a92aae4f48f01581ec5, VirusShare_c9c3be4d4651a01cef56d2c8b6922b36.

Table 4-6 shows all the feature values of these six ELF files. As we can see in Table 4-7 that we get accuracy of **0.93** for gini criterion.

Table 4-6. Decision tree with **gini** criterion for system call features value= [1, 5] feature values.

file	benign ma	execve	read	close	write	connect	chdir	bind	unlink	dup2	dup	socket	kill	rename	pipe	accept	creat	fork	open	clone
hciconfig	0	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
ef30be17cdc69e67d2c	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
ELF_8F83A5899F8D7C	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
VirusShare_391c74e62	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
VirusShare_a3f194088	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
VirusShare_c9c3be4d4	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0



(a). (b).

Figure 4-6. Decision tree with **gini** criterion analyzed by sandbox for system call features. (a) confusion matrix (b) ROC AUC graph.

Table 4-7. Decision tree model performance with **gini** criterion for system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.88	0.87	0.87	0.869	421
Malicious	0.95	0.95	0.95	0.949	1007
macro avg	0.91	0.91	0.91	**0.93**	1428
weighted avg	0.93	0.93	0.93	**0.93**	1428

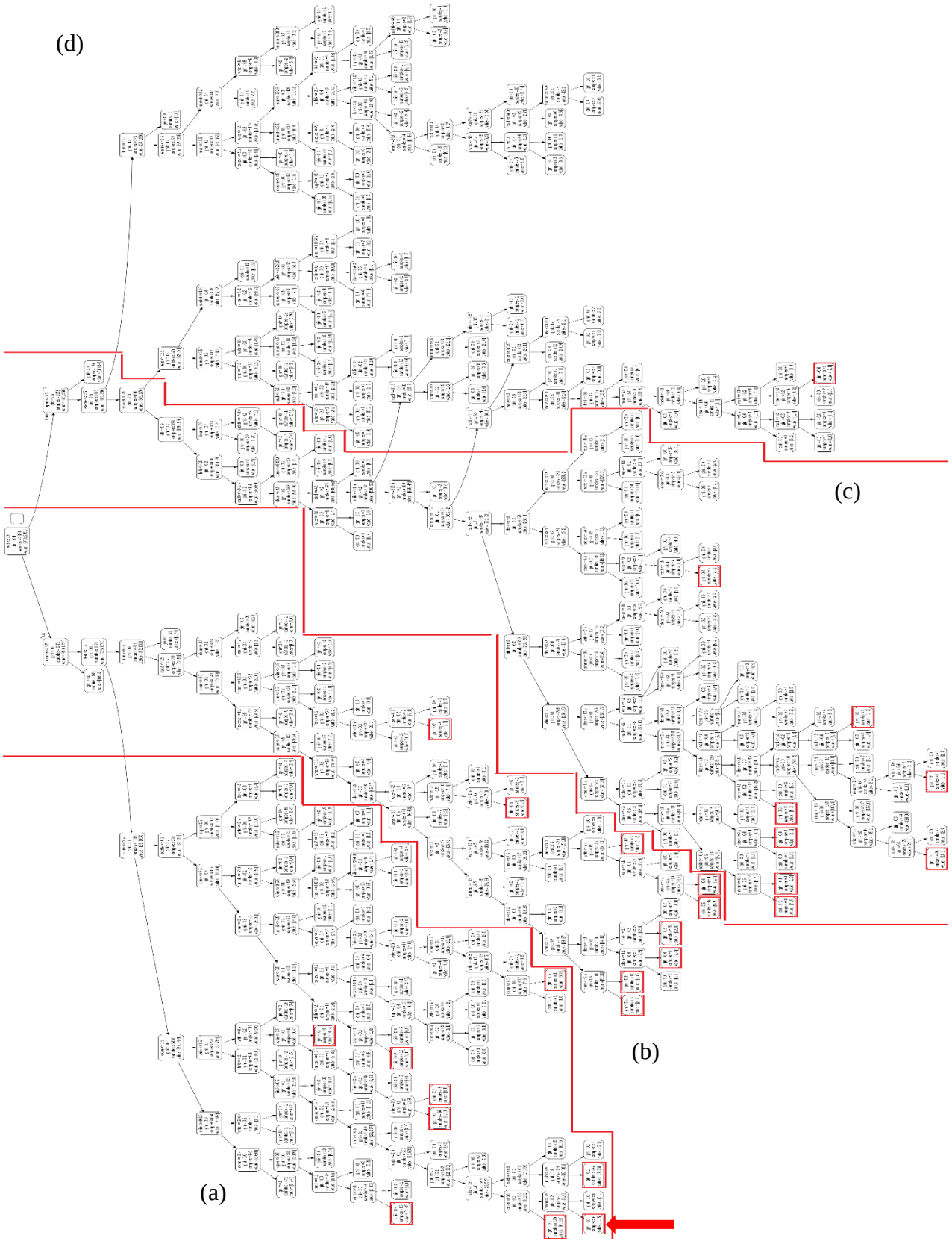


Figure 4-7. Decision tree with **gini** criterion analyzed by sandbox for system call features plot tree.

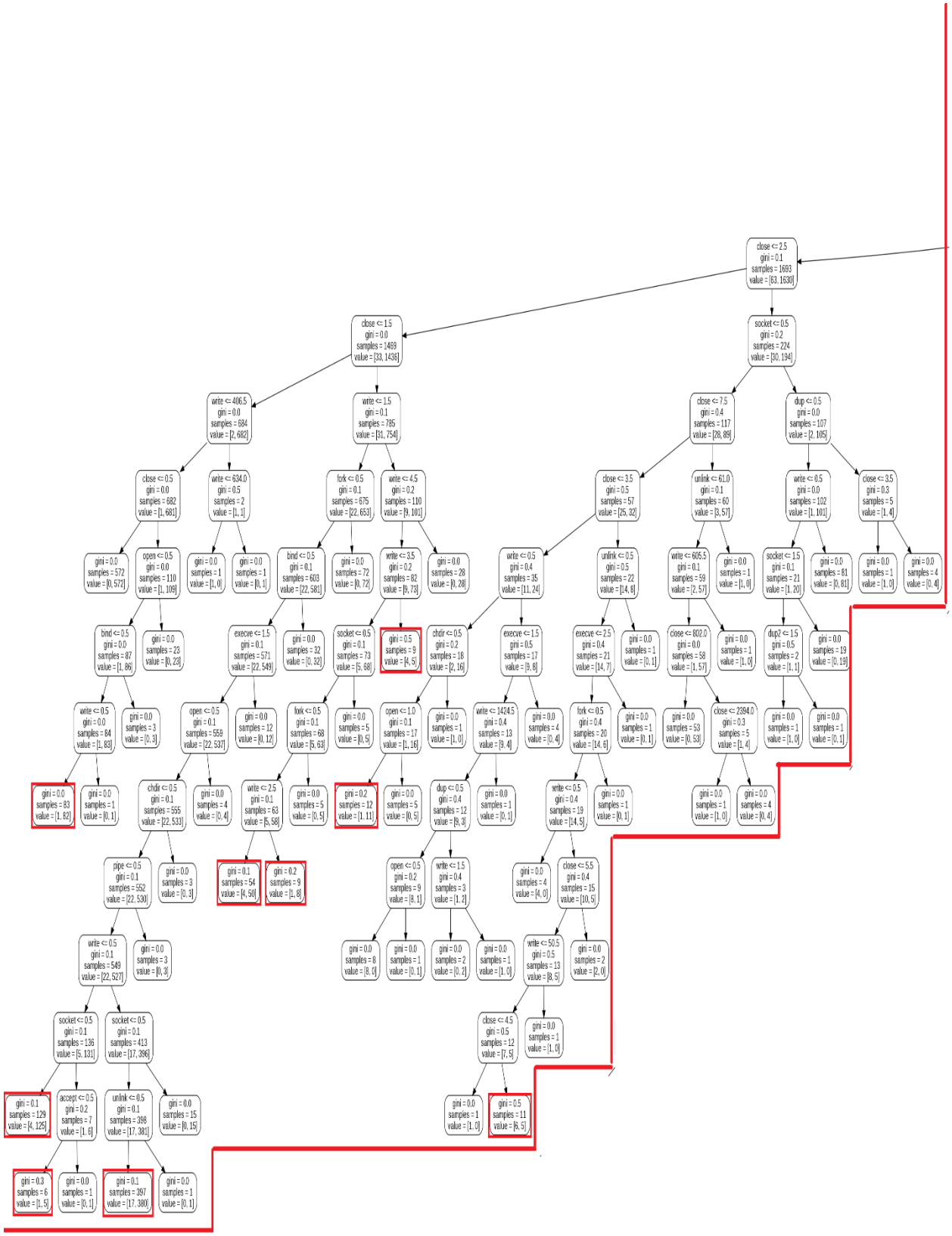

close <= 2.5
gini = 0.1
samples = 1693
value = [63, 1630]
close <= 1.5
gini = 0.0
samples = 1469
value = [33, 1436]
socket <= 0.5
gini = 0.2
samples = 224
value = [30, 194]
write <= 406.5
gini = 0.0
samples = 684
value = [2, 682]
write <= 1.5
gini = 0.1
samples = 785
value = [31, 754]
close <= 7.5
gini = 0.4
samples = 117
value = [28, 89]
dup <= 0.5
gini = 0.0
samples = 107
value = [2, 105]
close <= 0.5
gini = 0.0
samples = 682
value = [1, 681]
write <= 634.0
gini = 0.5
samples = 2
value = [1, 1]
fork <= 0.5
gini = 0.1
samples = 675
value = [22, 653]
write <= 4.5
gini = 0.2
samples = 110
value = [9, 101]
close <= 3.5
gini = 0.5
samples = 57
value = [25, 32]
unlink <= 61.0
gini = 0.1
samples = 60
value = [3, 57]
write <= 0.5
gini = 0.0
samples = 102
value = [1, 101]
close <= 3.5
gini = 0.3
samples = 5
value = [1, 4]
gini = 0.0
samples = 572
value = [0, 572]
open <= 0.5
gini = 0.0
samples = 110
value = [1, 109]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 1]
bind <= 0.5
gini = 0.1
samples = 603
value = [22, 581]
gini = 0.0
samples = 72
value = [0, 72]
write <= 3.5
gini = 0.2
samples = 82
value = [9, 73]
gini = 0.0
samples = 28
value = [0, 28]
write <= 0.5
gini = 0.4
samples = 35
value = [11, 24]
unlink <= 0.5
gini = 0.5
samples = 22
value = [14, 8]
write <= 605.5
gini = 0.1
samples = 59
value = [2, 57]
gini = 0.0
samples = 1
value = [1, 0]
socket <= 1.5
gini = 0.1
samples = 21
value = [1, 20]
gini = 0.0
samples = 81
value = [0, 81]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 4
value = [0, 4]
bind <= 0.5
gini = 0.0
samples = 87
value = [1, 86]
gini = 0.0
samples = 23
value = [0, 23]
execve <= 1.5
gini = 0.1
samples = 571
value = [22, 549]
gini = 0.0
samples = 32
value = [0, 32]
socket <= 0.5
gini = 0.1
samples = 73
value = [5, 68]
gini = 0.5
samples = 9
value = [4, 5]
chdir <= 0.5
gini = 0.2
samples = 18
value = [2, 16]
execve <= 1.5
gini = 0.5
samples = 17
value = [9, 8]
execve <= 2.5
gini = 0.4
samples = 21
value = [14, 7]
gini = 0.0
samples = 1
value = [0, 1]
close <= 802.0
gini = 0.0
samples = 58
value = [1, 57]
gini = 0.0
samples = 1
value = [1, 0]
dup2 <= 1.5
gini = 0.5
samples = 2
value = [1, 1]
gini = 0.0
samples = 19
value = [0, 19]
write <= 0.5
gini = 0.0
samples = 84
value = [1, 83]
gini = 0.0
samples = 3
value = [0, 3]
open <= 0.5
gini = 0.1
samples = 559
value = [22, 537]
gini = 0.0
samples = 12
value = [0, 12]
fork <= 0.5
gini = 0.1
samples = 68
value = [5, 63]
gini = 0.0
samples = 5
value = [0, 5]
open <= 1.0
gini = 0.1
samples = 17
value = [1, 16]
gini = 0.0
samples = 1
value = [1, 0]
write <= 1424.5
gini = 0.4
samples = 13
value = [9, 4]
gini = 0.0
samples = 4
value = [0, 4]
fork <= 0.5
gini = 0.4
samples = 20
value = [14, 6]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 53
value = [0, 53]
close <= 2394.0
gini = 0.3
samples = 5
value = [1, 4]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 83
value = [1, 82]
gini = 0.0
samples = 1
value = [0, 1]
chdir <= 0.5
gini = 0.1
samples = 555
value = [22, 533]
gini = 0.0
samples = 4
value = [0, 4]
write <= 2.5
gini = 0.1
samples = 63
value = [5, 58]
gini = 0.0
samples = 5
value = [0, 5]
gini = 0.2
samples = 12
value = [1, 11]
gini = 0.0
samples = 5
value = [0, 5]
dup <= 0.5
gini = 0.4
samples = 12
value = [9, 3]
gini = 0.0
samples = 1
value = [0, 1]
write <= 0.5
gini = 0.4
samples = 19
value = [14, 5]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 4
value = [0, 4]
pipe <= 0.5
gini = 0.1
samples = 552
value = [22, 530]
gini = 0.0
samples = 3
value = [0, 3]
gini = 0.1
samples = 54
value = [4, 50]
gini = 0.2
samples = 9
value = [1, 8]
open <= 0.5
gini = 0.2
samples = 9
value = [8, 1]
write <= 1.5
gini = 0.4
samples = 3
value = [1, 2]
gini = 0.0
samples = 4
value = [4, 0]
close <= 5.5
gini = 0.4
samples = 15
value = [10, 5]
write <= 0.5
gini = 0.1
samples = 549
value = [22, 527]
gini = 0.0
samples = 3
value = [0, 3]
gini = 0.0
samples = 8
value = [8, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 1
value = [1, 0]
write <= 50.5
gini = 0.5
samples = 13
value = [8, 5]
gini = 0.0
samples = 2
value = [2, 0]
socket <= 0.5
gini = 0.1
samples = 136
value = [5, 131]
socket <= 0.5
gini = 0.1
samples = 413
value = [17, 396]
close <= 4.5
gini = 0.5
samples = 12
value = [7, 5]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.1
samples = 129
value = [4, 125]
accept <= 0.5
gini = 0.2
samples = 7
value = [1, 6]
unlink <= 0.5
gini = 0.1
samples = 398
value = [17, 381]
gini = 0.0
samples = 15
value = [0, 15]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.5
samples = 11
value = [6, 5]
gini = 0.3
samples = 6
value = [1, 5]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.1
samples = 397
value = [17, 380]
gini = 0.0
samples = 1
value = [0, 1]

Figure 4-7. (a)

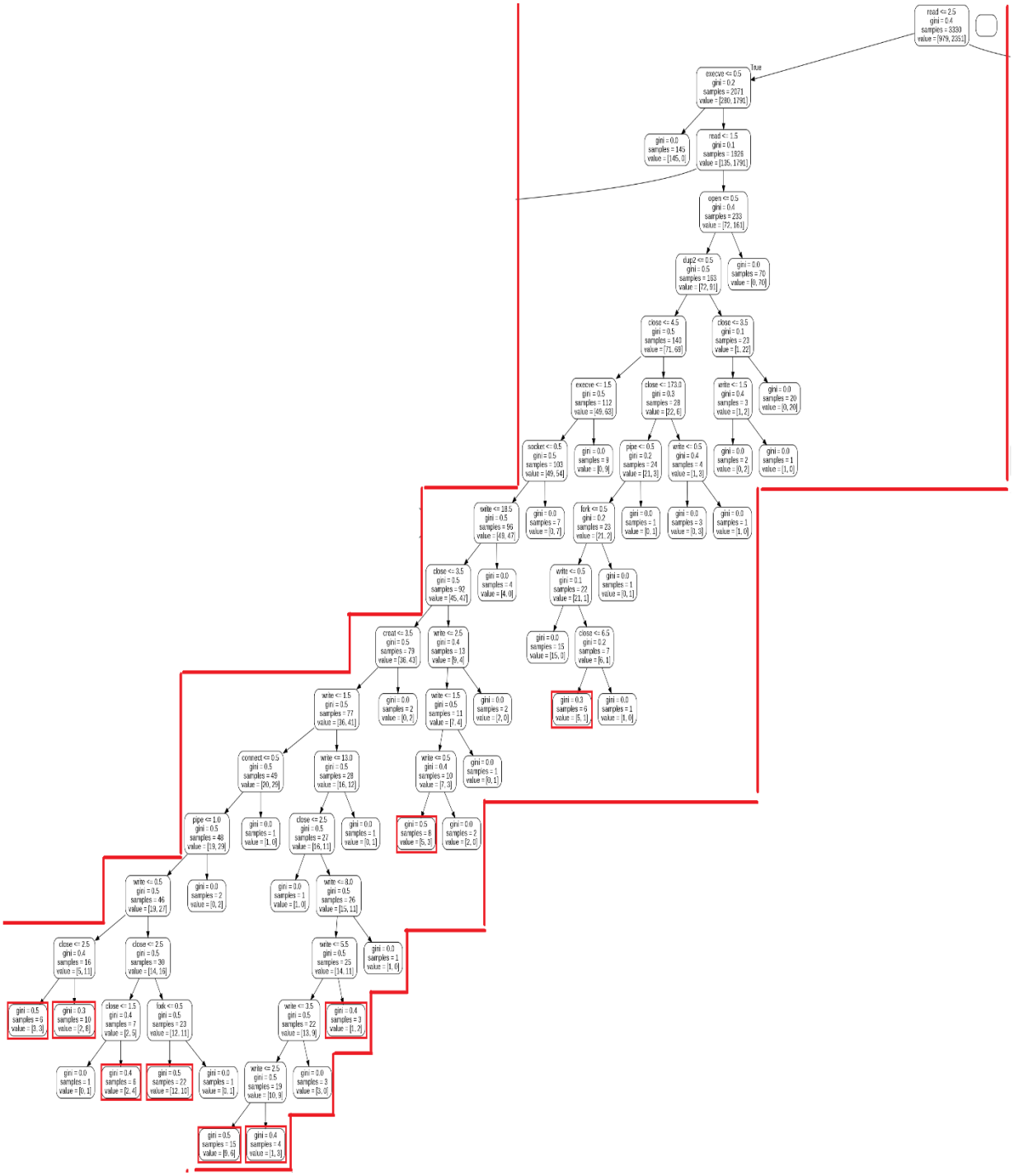


Figure 4-7. (b)

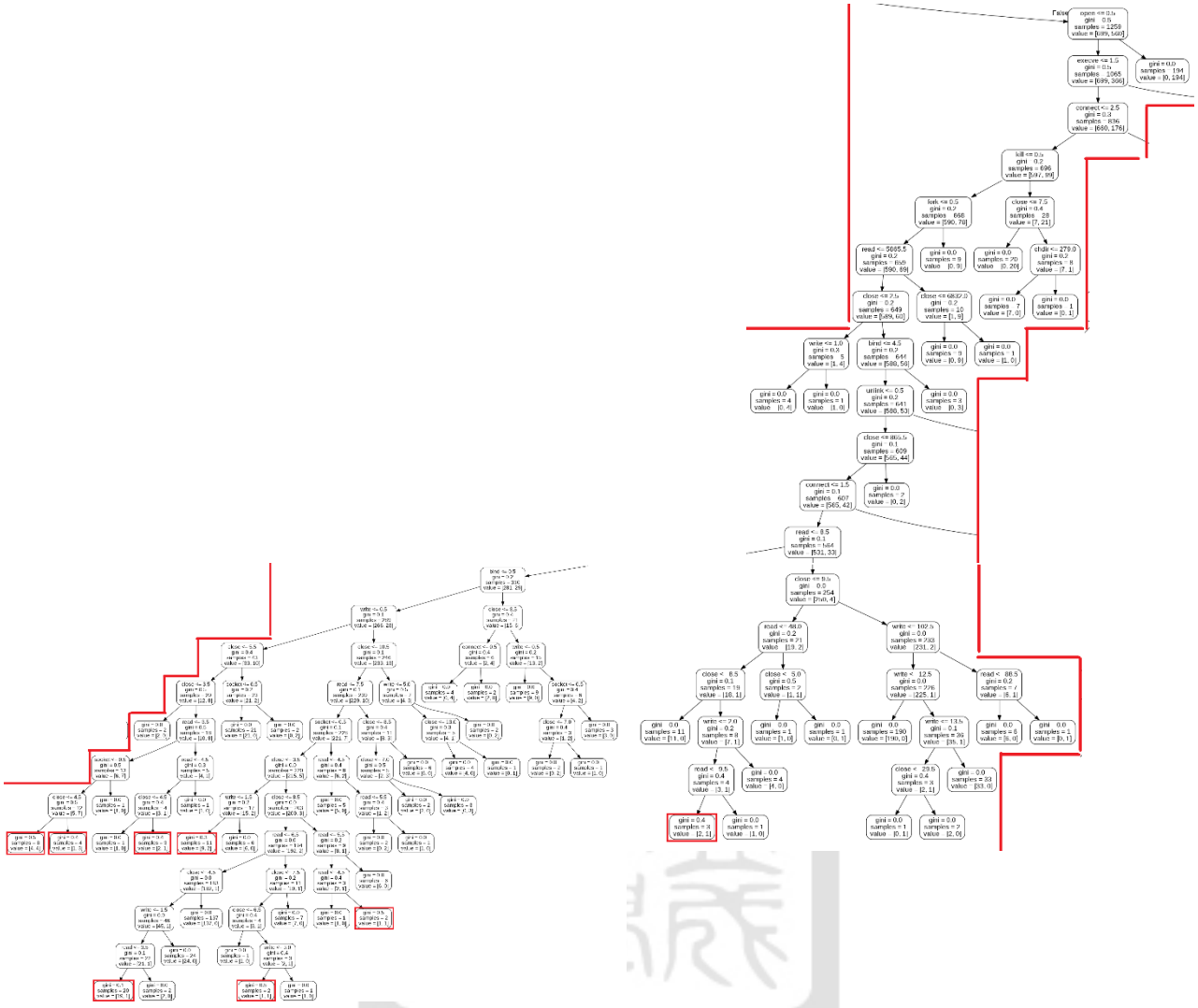


Figure 4-7. (c)

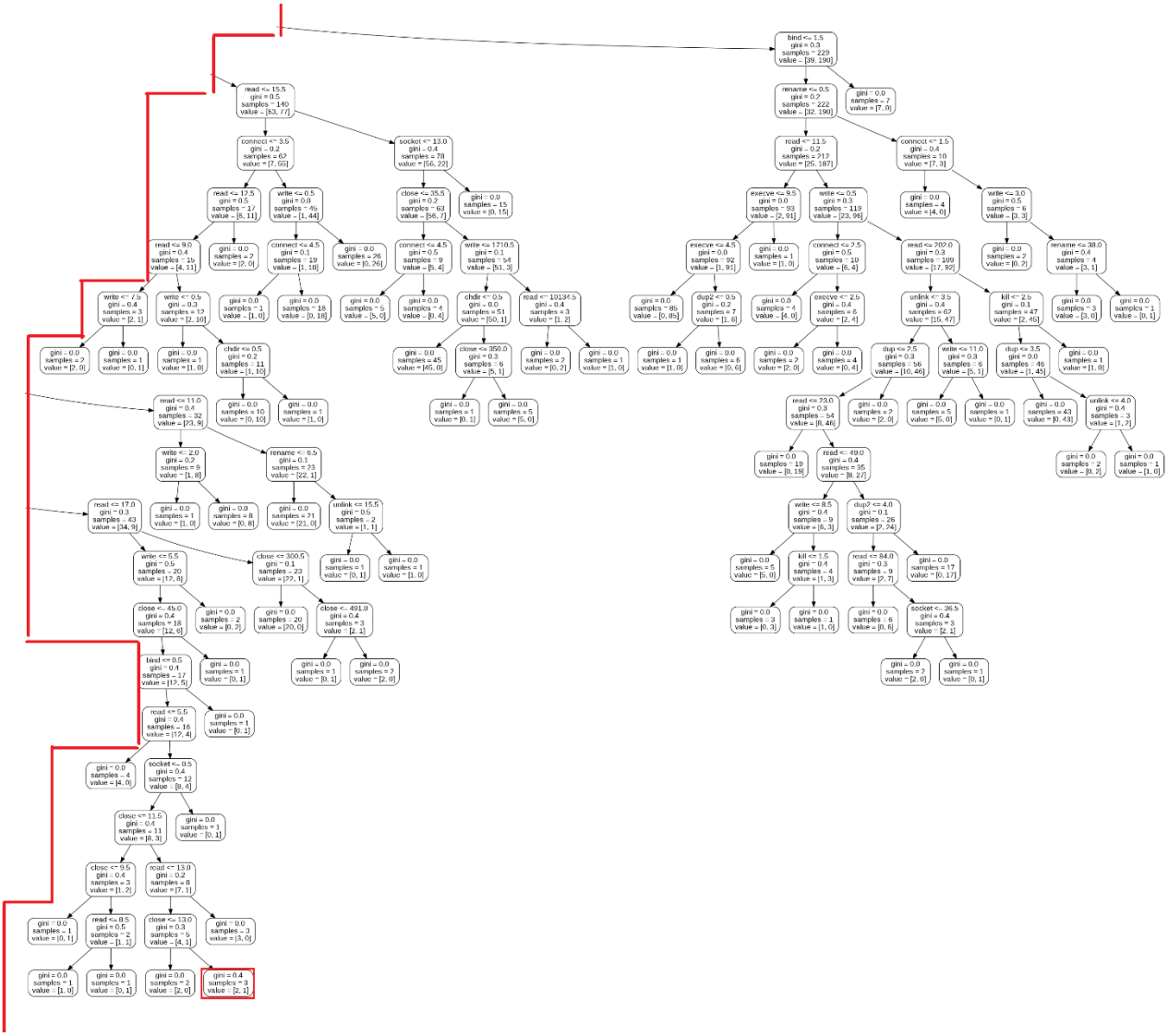
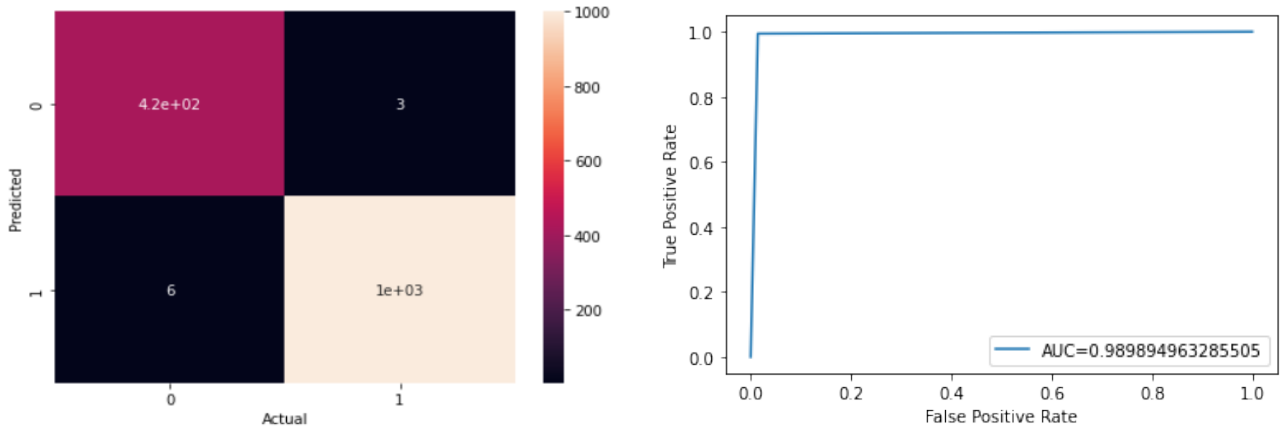


Figure 4-7. (d)



(a). (b).

Figure 4-8. Decision tree with **gini** criterion analyzed by sandbox for combined features. (a) confusion matrix (b) ROC AUC graph.

Table 4-8. Decision tree model performance with **gini** criterion for combined features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.99	0.99	0.99	0.993	421
Malicious	1.00	0.99	1.00	0.994	1007
macro avg	0.99	0.99	0.99	**0.99**	1428
weighted avg	0.99	0.99	0.99	**0.99**	1428

For combined features, our results using gini criterion is shown in Figure 4-8, and Table 4-8. In Figure 4-8(b), we can see that AUC=0.990. In Table 4-8,

Benign-precision=418/(418+6)=0.986, Benign-recall=418/(418+3)=0.993,

Malicious-precision=1001/(3+1001)=0.997, Malicious-recall=1001/(6+1001)=0.994,

macroavg-precision=(0.99+1.00)/2=0.99, macroavg-recall=(0.99+0.99)/2=0.99,

weightedavg-precision=(0.99*421+1.00*1007)/1428=0.99,

weightedavg-recall=(0.99*421+0.99*1007)/1428=0.99,

Benign-f1score=2*0.986*0.993/(0.986+0.993)=1.958196/1.979=0.989,

Malicious-f1score=2*0.997*0.994/(0.997+0.994)=1.982036/1.991=0.995. The detailed decision trees built by our programs is also shown in Figure 4-9. As we can see in Table 4-8 that we get accuracy of **0.99** for gini criterion.

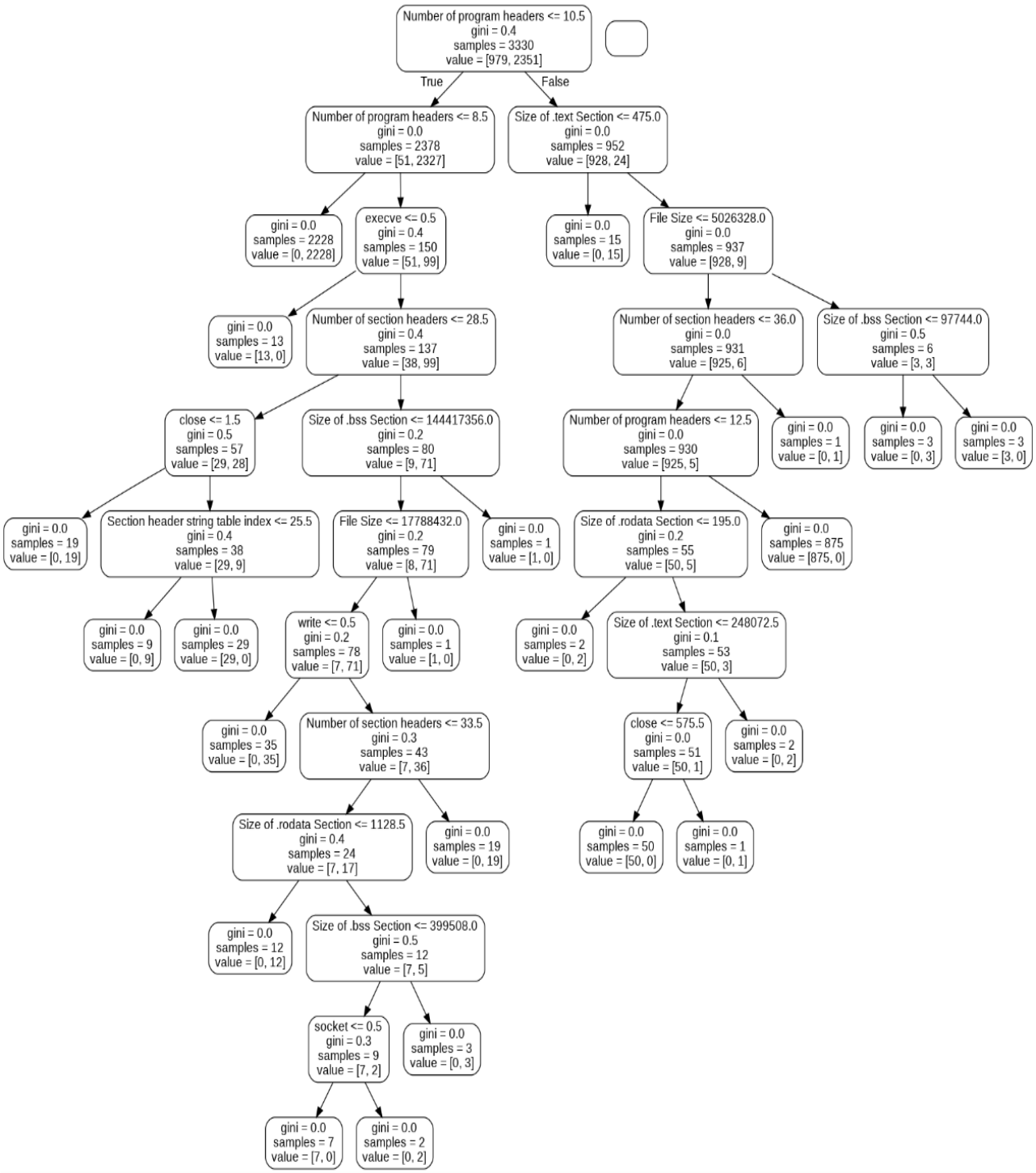
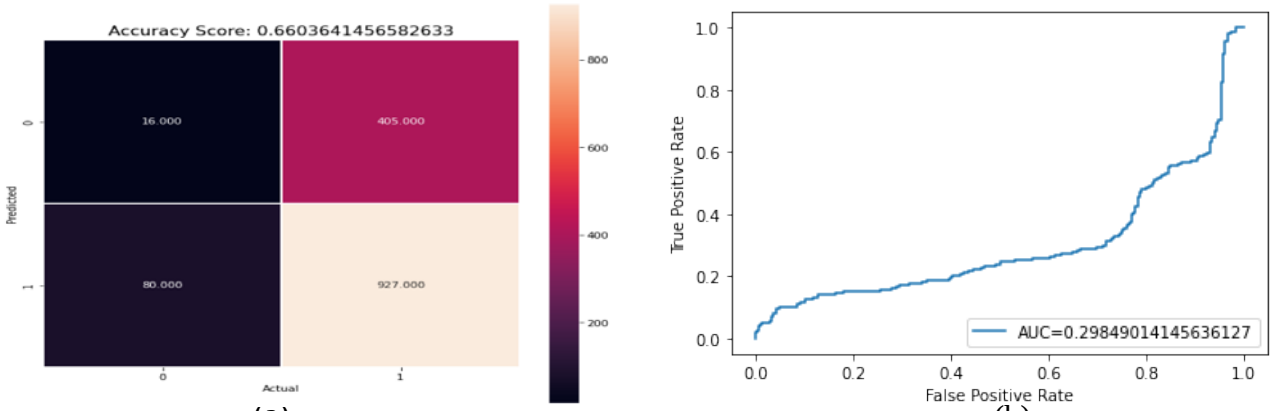


Figure 4-9. Decision tree with **gini** criterion analyzed by sandbox for combined features plot tree.

4.2.2 Logistic Regression

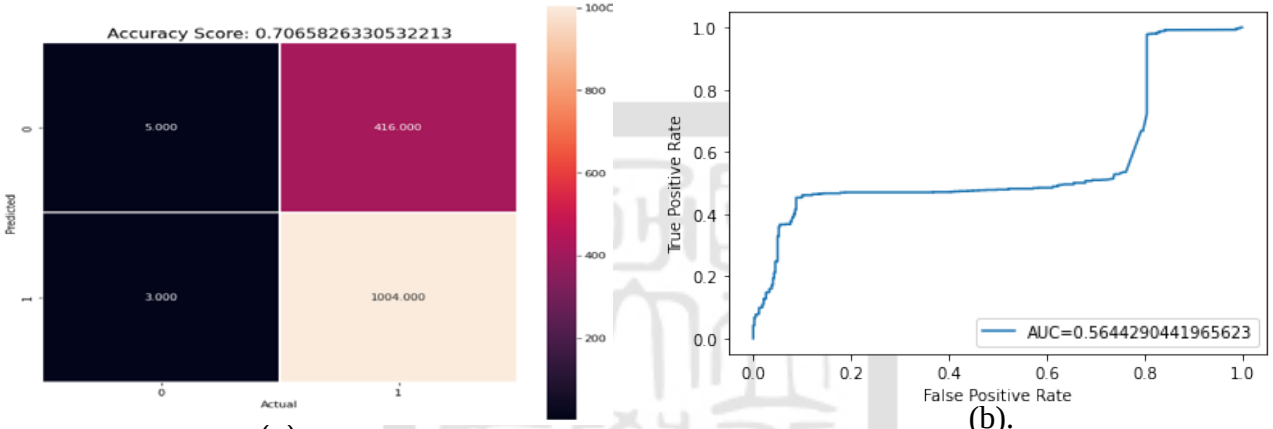
We examine the results of logistic regression model trained individually on the static, system call, and combined features. For static features, our results are shown in Figure 4-10, and Table 4-9. In Figure 4-10(b), we can see that AUC=0.298. In Table 4-9, Benign-precision=16/(16+80)=0.167, Benign-recall=16/(16+405)=0.038,
Malicious-precision=927/(405+927)=0.696, Malicious-recall=927/(80+927)=0.921,
macroavg-precision=(0.17+0.70)/2=0.43, macroavg-recall=(0.04+0.92)/2=0.48,
weightedavg-precision=(0.17*421+0.70*1007)/1428=0.54,
weightedavg-recall=(0.04*421+0.92*1007)/1428=0.66,
Benign-f1score=2*0.167*0.038/(0.167+0.038)=0.012692/0.205=0.062,
Malicious-f1score=2*0.696*0.921/(0.696+0.921)=1.282032/1.617=0.793. As we can see in Table 4-9 that we get accuracy of **0.66**.
For system call features, our results are shown in Figure 4-11, and Table 4-10. In Figure 4-11(b), we can see that AUC=0.564. In Table 4-10,
Benign-precision=5/(5+3)=0.625, Benign-recall=5/(5+416)=0.012,
Malicious-precision=1004/(416+1004)=0.707, Malicious-recall=1004/(3+1004)=0.997,
macroavg-precision=(0.62+0.71)/2=0.67, macroavg-recall=(0.01+1.00)/2=0.50,
weightedavg-precision=(0.62*421+0.71*1007)/1428=0.68,
weightedavg-recall=(0.01*421+1.00*1007)/1428=0.71,
Benign-f1score=2*0.625*0.012/(0.625+0.012)=0.015/0.637=0.024,
Malicious-f1score=2*0.707*0.997/(0.707+0.997)=1.409758/1.704=0.827. As we can see in Table 4-10 that we get accuracy of **0.71**.
For combined features, our results are shown in Figure 4-12, and Table 4-11. In Figure 4-12(b), we can see that AUC=0.306. In Table 4-11,

Benign-precision=14/(14+66)=0.175, Benign-recall=14/(14+407)=0.033,



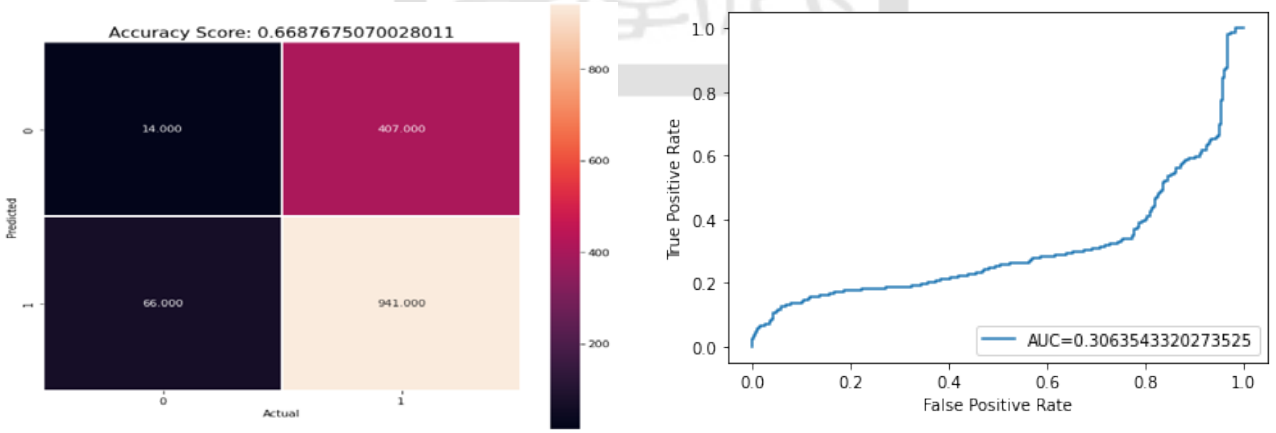
(a). (b).

Figure 4-10. Logistic regression analyzed by sandbox for static features. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-11. Logistic regression analyzed by sandbox for system call features. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-12. Logistic regression analyzed by sandbox for combined features. (a) confusion matrix (b) ROC AUC graph.

Malicious-precision=941/(407+941)=0.698, Malicious-recall=941/(66+941)=0.934,

macroavg-precision=(0.17+0.70)/2=0.44, macroavg-recall=(0.03+0.93)/2=0.48,

weightedavg-precision=(0.17*421+0.70*1007)/1428=0.54,

Table 4-9. Logistic regression model performance for static features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.17	0.04	0.06	0.038	421
Malicious	0.70	0.92	0.79	0.921	1007
macro avg	0.43	0.48	0.43	**0.66**	1428
weighted avg	0.54	0.66	0.58	**0.66**	1428

Table 4-10. Logistic regression model performance for system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.62	0.01	0.02	0.012	421
Malicious	0.71	1.00	0.83	0.997	1007
macro avg	0.67	0.50	0.43	**0.71**	1428
weighted avg	0.68	0.71	0.59	**0.71**	1428

Table 4-11. Logistic regression model performance for combined features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.17	0.03	0.06	0.033	421
Malicious	0.70	0.93	0.80	0.934	1007
macro avg	0.44	0.48	0.43	**0.67**	1428
weighted avg	0.54	0.67	0.58	**0.67**	1428

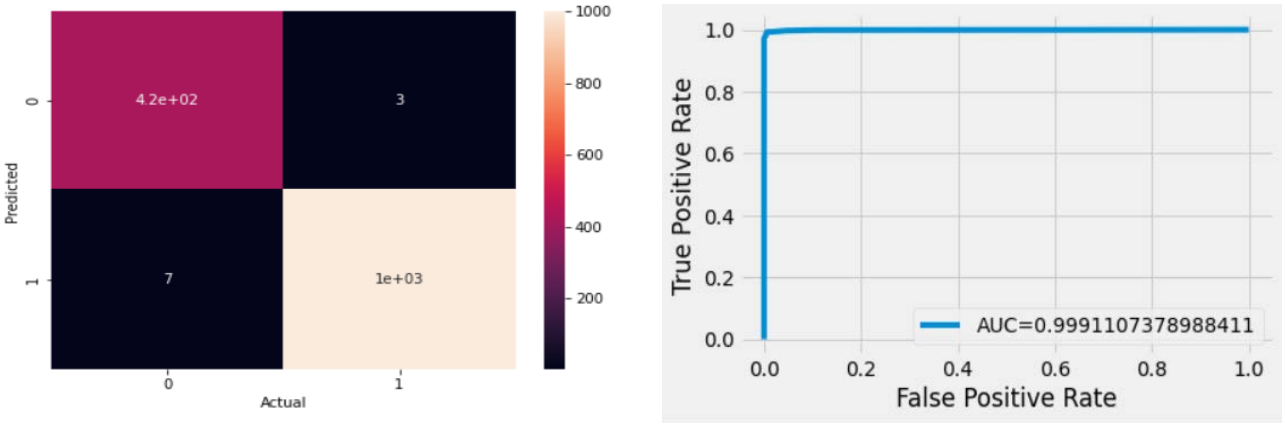
weightedavg-recall=(0.03*421+0.93*1007)/1428=0.67,

Benign-f1score=2*0.175*0.033/(0.175+0.033)=0.01155/0.208=0.056,

Malicious-f1score=2*0.698*0.934/(0.698+0.934)=1.303864/1.632=0.799. As we can see in Table 4-11 that we get accuracy of **0.67**. We found that logistic regression model performances are all worse than decision tree model performance.

4.2.3 Random Forest

We examine the results of random forest model with gini criterions trained individually on the static, system call, and combined features. For static features, our results using gini criterion is shown in Figure 4-13 and Table 4-12. In Figure 4-13(b), we can see that AUC=0.999. In Table 4-12, Benign-precision=418/(418+7)=0.984, Benign-recall=418/(418+3)=0.993, Malicious-precision=1000/(3+1000)=0.997, Malicious-recall=1000/(7+1000)=0.993, macroavg-precision=(0.98+1.00)/2=0.99, macroavg-recall=(0.99+0.99)/2=0.99, weightedavg-precision=(0.98*421+1.00*1007)/1428=0.99, weightedavg-recall=(0.99*421+0.99*1007)/1428=0.99, Benign-f1score=2*0.984*0.993/(0.984+0.993)=1.954224/1.977=0.988, Malicious-f1score=2*0.997*0.993/(0.997+0.993)=1.980042/1.99=0.995. The variable importance is shown in Figure 4-14. We found that feature “Number of program headers” is the most important variable, so we choose this feature as our tree’s root. We set *n_estimators* to 10 to generate 10 trees in this forest. We selected the 9th tree with “Number of program headers” as the root. The detailed 9th of the trees in the random forest built by our program is also shown in Figure 4-15. As we can see in Table 4-11 that we get accuracy of **0.99** for gini criterion.



(a). (b).

Figure 4-13. Random forest with **gini** criterion analyzed by sandbox for static features. (a) confusion matrix (b) ROC AUC graph.

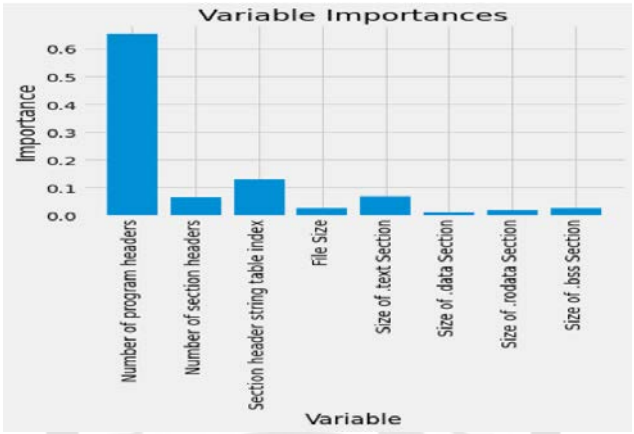


Figure 4-14. Random forest with **gini** criterion analyzed by sandbox variable importance for static features.

Table 4-12. Random forest with gini criterion model performance for static features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.98	0.99	0.99	0.993	421
Malicious	1.00	0.99	1.00	0.993	1007
macro avg	0.99	0.99	0.99	**0.99**	1428
weighted avg	0.99	0.99	0.99	**0.99**	1428

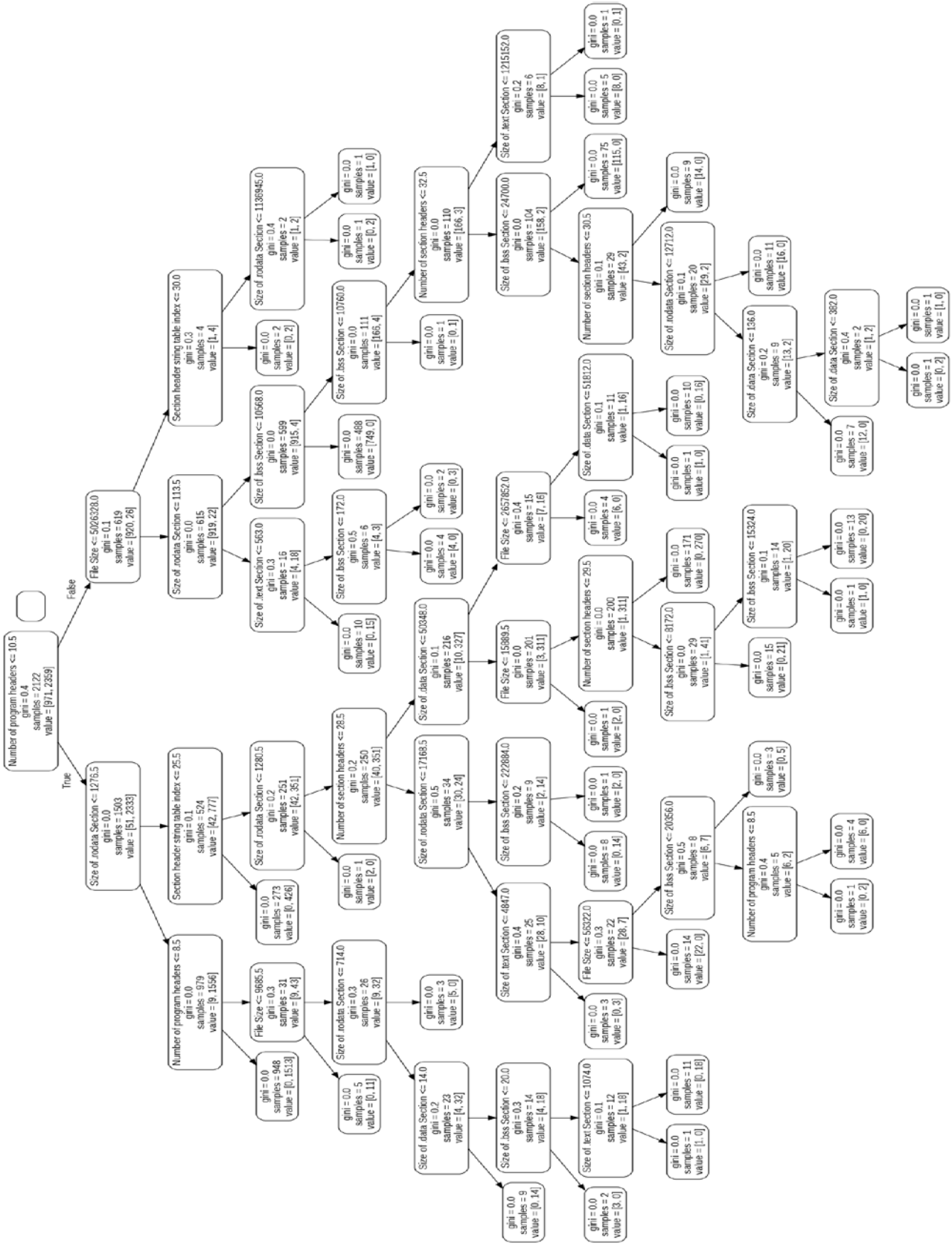
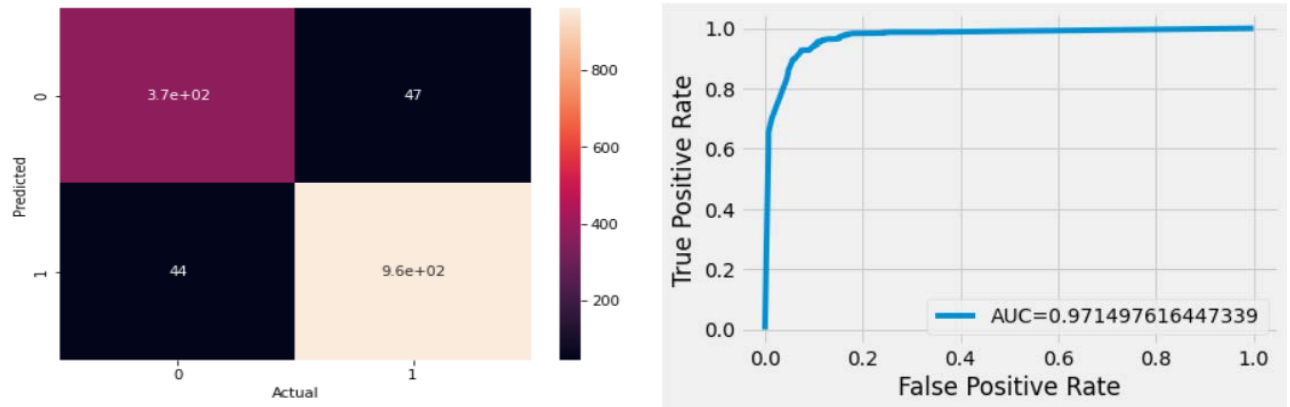


Figure 4-15. Random forest with **gini** criterion analyzed by sandbox for static features plot tree.

For system call features, our results using gini criterion is shown in Figure 4-16 and Table 4-14. In Figure 4-16(b), we can see that AUC=0.971. In Table 4-14, Benign-precision=374/(374+44)=0.895, Benign-recall=374/(374+47)=0.888, Malicious-precision=963/(47+963)=0.953, Malicious-recall=963/(44+963)=0.956, macroavg-precision=(0.89+0.95)/2=0.92, macroavg-recall=(0.89+0.96)/2=0.92, weightedavg-precision=(0.89*421+0.95*1007)/1428=0.94, weightedavg-recall=(0.89*421+0.96*1007)/1428=0.94, Benign-f1score=2*0.895*0.888/(0.895+0.888)=1.58952/1.783=0.891, Malicious-f1score=2*0.953*0.956/(0.953+0.956)=1.822136/1.909=0.954. The variable importance is shown in Figure 4-17, we found that feature “execve” is the most important variable, so we choose feature “execve” as our tree’s root. We set *n_estimators* to 20, which means that there are 20 trees in this forest, and we selected the 16th tree with “execve” as the root. The detailed one of the trees in the random forest built by our program is also shown in Figure 4-18. From Figure 4-18, it can be found that there is still no way to completely distinguish between benign files and malicious files in the value = [benign, malicious] in the red box of the leaf node, so the accuracy can only reach **0.94**. Take value= [1, 4] as an example. One benign file, hciconfig, cannot be distinguished. Five malicious files that cannot be distinguished are ELF_8F83A5899F8D7C93BEA65F8BC7BCF25C, ef30be17cdc69e67d2c82ac2e002190698e337595bf42cbe65e10f5794fa6de2, ELF_8F83A5899F8D7C93BEA65F8BC7BCF25C, VirusShare_391c74e627dfea7d3a5e1e9ba807a2d8, VirusShare_a3f194088d346a92aae4f48f01581ec5. Table 4-13 shows all the feature values of these five ELF files. As we can see in Table 4-14 that we get accuracy of **0.94** for gini criterion.

Table 4-13. Random forest with **gini** criterion for system call features value= [1, 4] feature values.

file	benign ma	execve	read	close	write	connect	chdir	bind	unlink	dup2	dup	socket	kill	rename	pipe	accept	creat	fork	open	clone
hciconfig	0	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
ef30be17c	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
ELF_8F83A	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
VirusShare	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
VirusShare	1	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0



(a). (b).

Figure 4-16. Random forest with **gini** criterion analyzed by sandbox for system call features. (a) confusion matrix (b) ROC AUC graph.

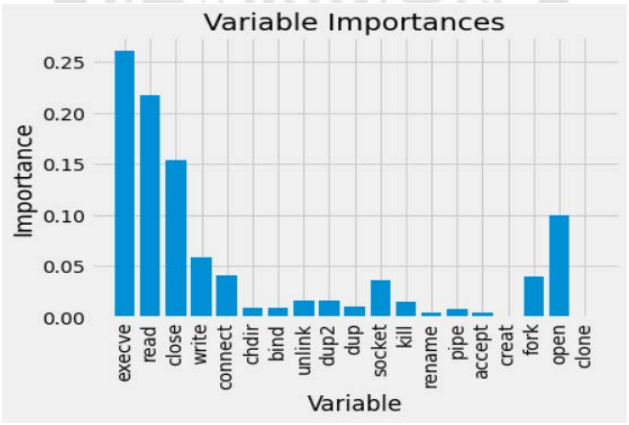


Figure 4-17. Random forest with **gini** criterion analyzed by sandbox variable importance for system call features.

Table 4-14. Random forest with gini criterion model performance for system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.89	0.89	0.89	0.888	421
Malicious	0.95	0.96	0.95	0.956	1007
macro avg	0.93	0.92	0.92	**0.94**	1428
weighted avg	0.94	0.94	0.94	**0.94**	1428

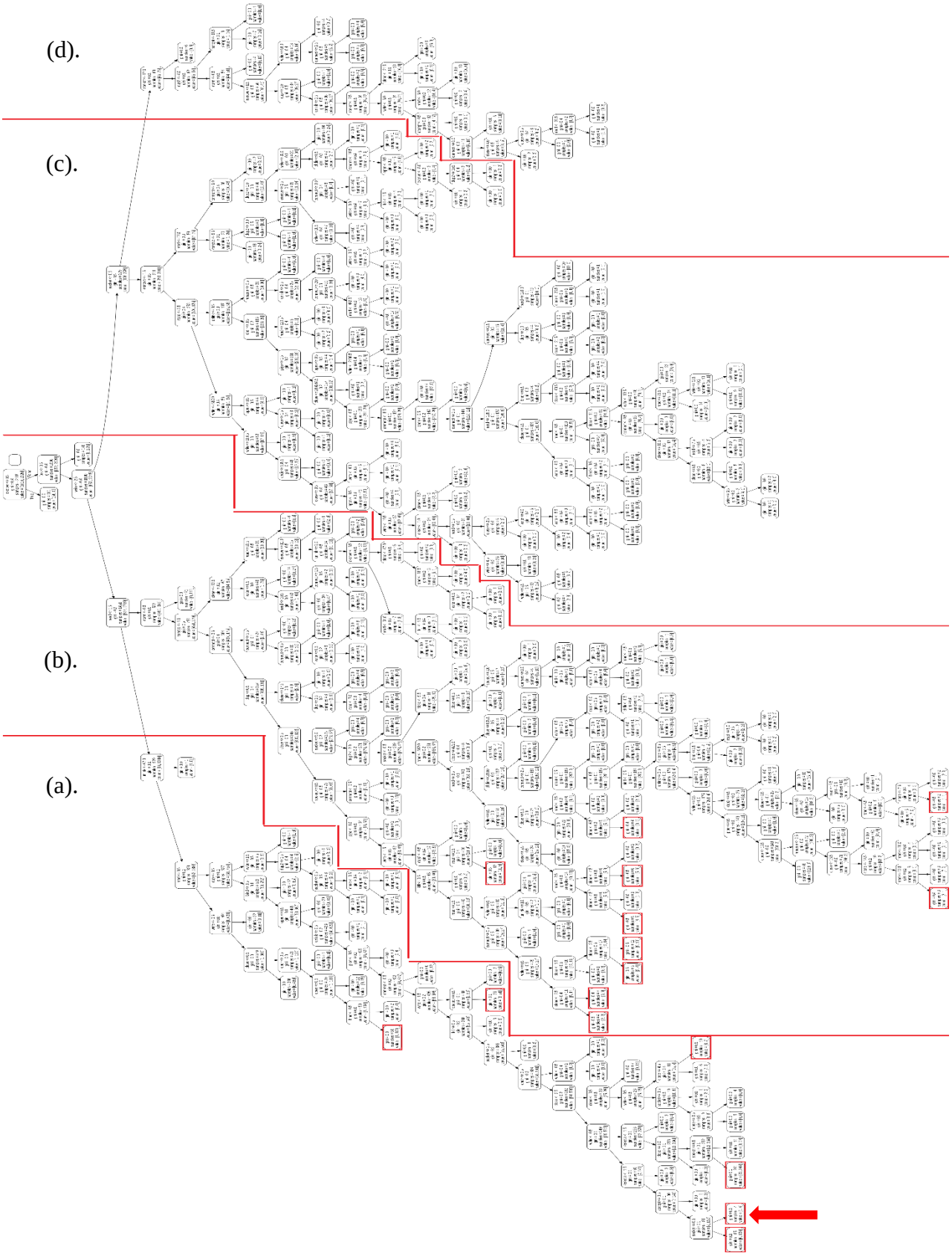


Figure 4-18. Random forest with **gini** criterion analyzed by sandbox for system call features plot tree.

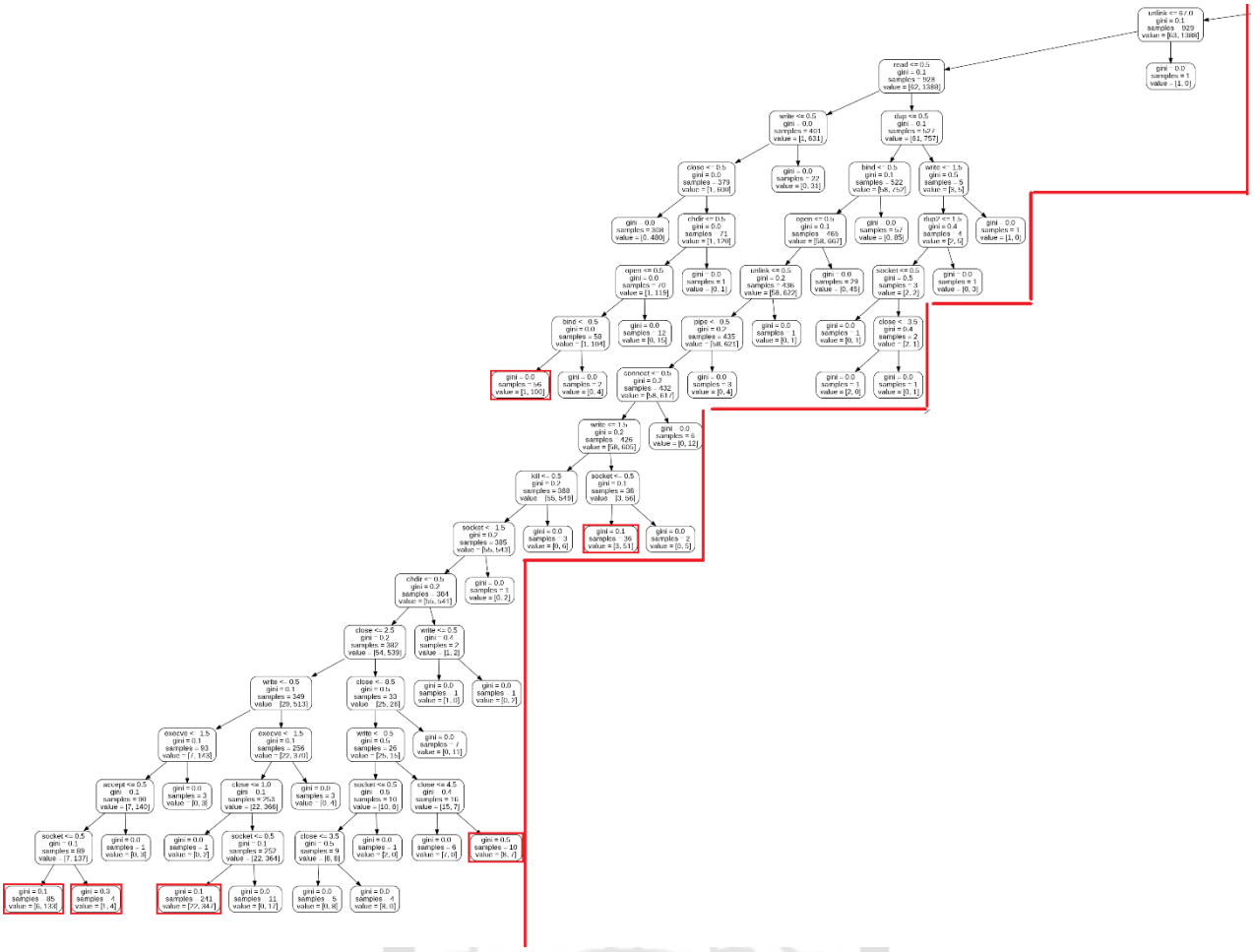


Figure 4-18. (a).

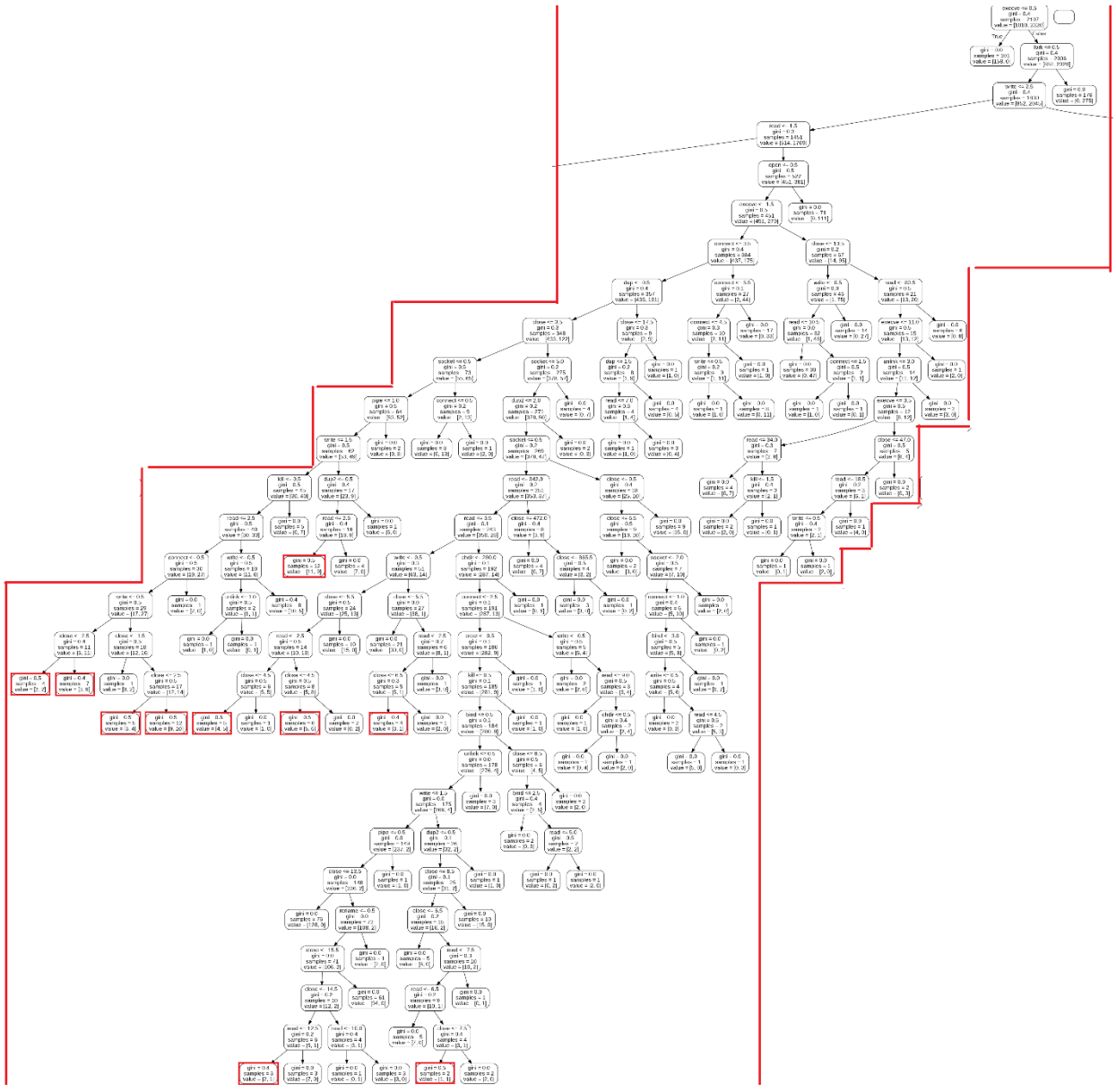


Figure 4-18. (b).

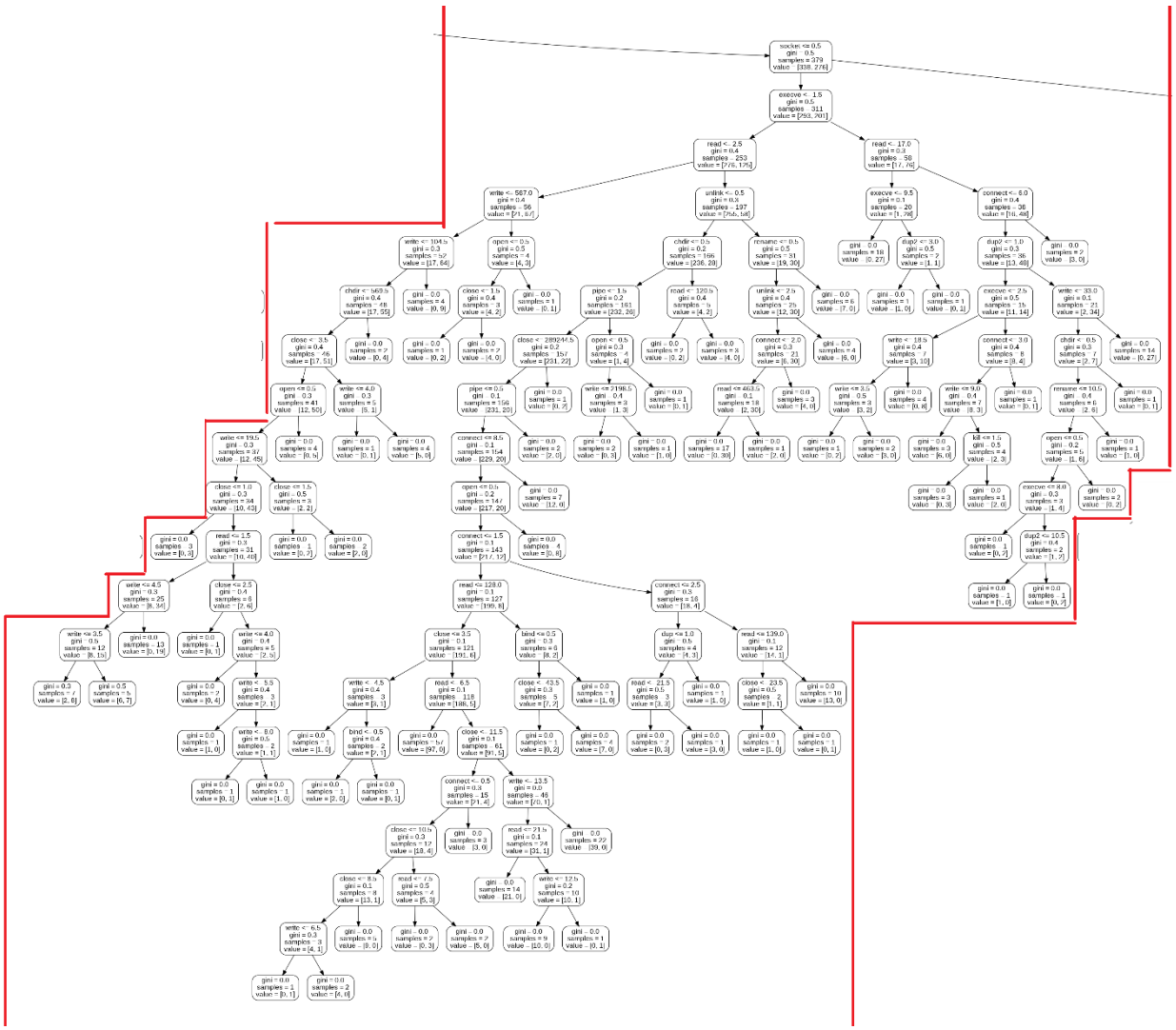


Figure 4-18. (c).

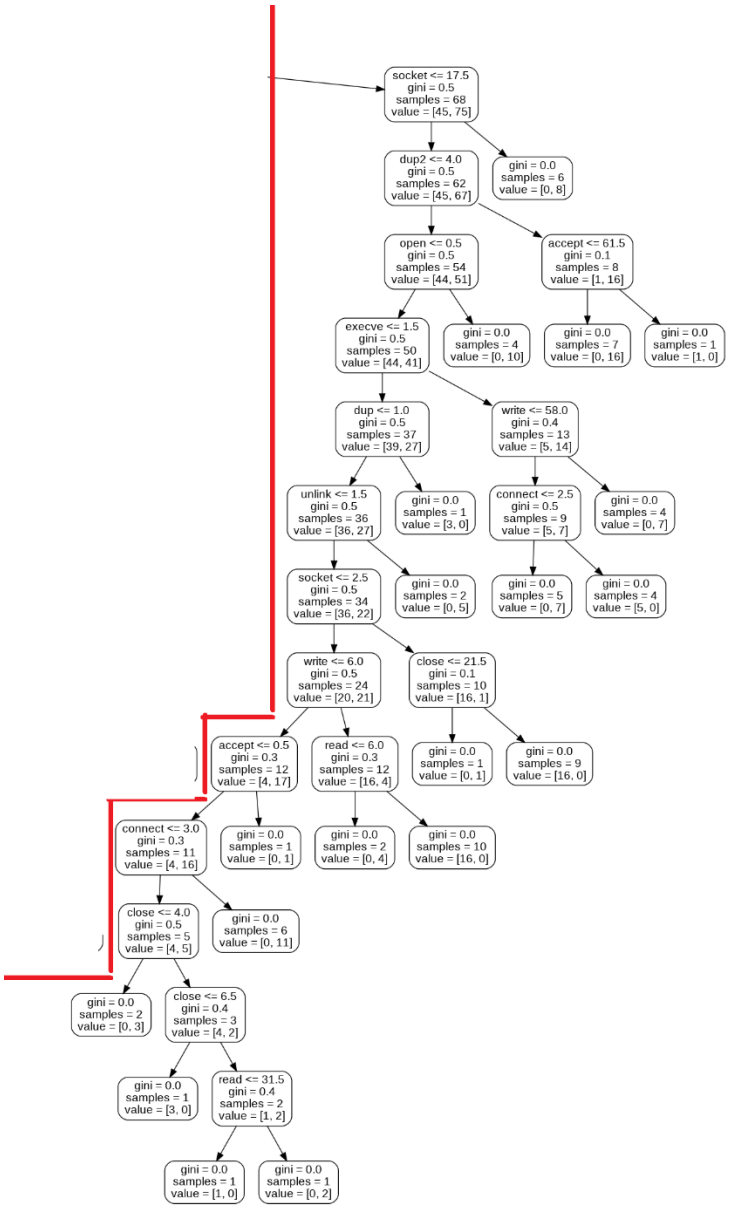
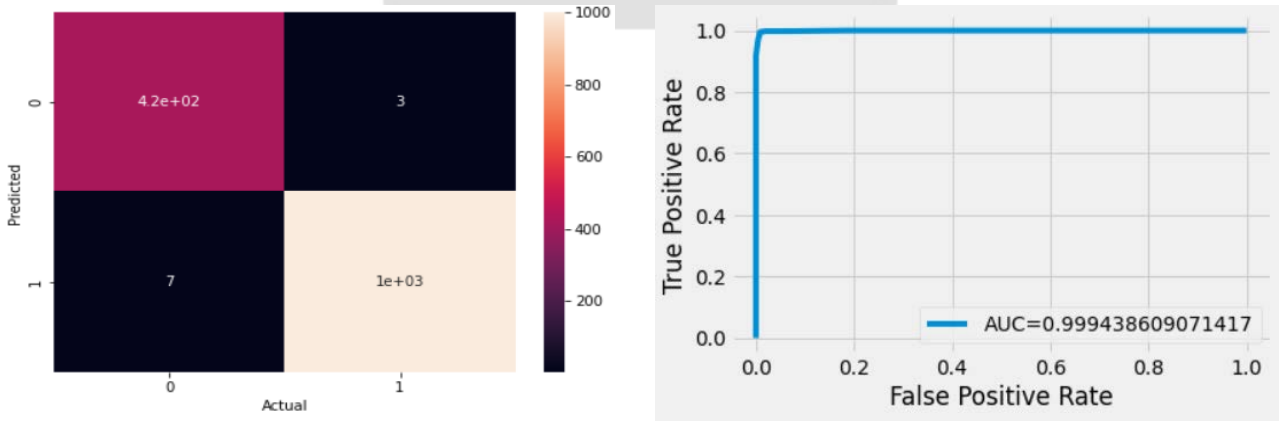

socket <= 17.5
gini = 0.5
samples = 68
value = [45, 75]
dup2 <= 4.0
gini = 0.5
samples = 62
value = [45, 67]
gini = 0.0
samples = 6
value = [0, 8]
open <= 0.5
gini = 0.5
samples = 54
value = [44, 51]
accept <= 61.5
gini = 0.1
samples = 8
value = [1, 16]
execve <= 1.5
gini = 0.5
samples = 50
value = [44, 41]
gini = 0.0
samples = 4
value = [0, 10]
gini = 0.0
samples = 7
value = [0, 16]
gini = 0.0
samples = 1
value = [1, 0]
dup <= 1.0
gini = 0.5
samples = 37
value = [39, 27]
write <= 58.0
gini = 0.4
samples = 13
value = [5, 14]
unlink <= 1.5
gini = 0.5
samples = 36
value = [36, 27]
gini = 0.0
samples = 1
value = [3, 0]
connect <= 2.5
gini = 0.5
samples = 9
value = [5, 7]
gini = 0.0
samples = 4
value = [0, 7]
socket <= 2.5
gini = 0.5
samples = 34
value = [36, 22]
gini = 0.0
samples = 2
value = [0, 5]
gini = 0.0
samples = 5
value = [0, 7]
gini = 0.0
samples = 4
value = [5, 0]
write <= 6.0
gini = 0.5
samples = 24
value = [20, 21]
close <= 21.5
gini = 0.1
samples = 10
value = [16, 1]
accept <= 0.5
gini = 0.3
samples = 12
value = [4, 17]
read <= 6.0
gini = 0.3
samples = 12
value = [16, 4]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 9
value = [16, 0]
connect <= 3.0
gini = 0.3
samples = 11
value = [4, 16]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 2
value = [0, 4]
gini = 0.0
samples = 10
value = [16, 0]
close <= 4.0
gini = 0.5
samples = 5
value = [4, 5]
gini = 0.0
samples = 6
value = [0, 11]
gini = 0.0
samples = 2
value = [0, 3]
close <= 6.5
gini = 0.4
samples = 3
value = [4, 2]
gini = 0.0
samples = 1
value = [3, 0]
read <= 31.5
gini = 0.4
samples = 2
value = [1, 2]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 2]

Figure 4-18. (d).

For combined features, our results using gini criterion are shown in Figure 4-19 and Table 4-15. In Figure 4-19(b), we can see that AUC=0.999. In Table 4-15, Benign-precision=418/(418+7)=0.984, Benign-recall=418/(418+3)=0.993, Malicious-precision=1000/(3+1000)=0.997, Malicious-recall=1000/(7+1000)=0.993, macroavg-precision=(0.98+1.00)/2=0.999, macroavg-recall=(0.99+0.99)/2=0.99, weightedavg-precision=(0.98*421+1.00*1007)/1428=0.99, weightedavg-recall=(0.99*421+0.99*1007)/1428=0.99, Benign-f1score=2*0.984*0.993/(0.984+0.993)=1.954224/1.977=0.988, Malicious-f1score=2*0.997*0.993/(0.997+0.993)=1.980042/1.99=0.995. The variable importance is shown in Figure 4-20. We found that feature “Number of program headers” is the most important variable, so we choose this feature as our tree’s root. We set *n_estimators* to 10in order to generate 10 trees in this forest. We selected the 9th tree with “Number of program headers” as the root. The detailed 9th of the trees in the random forest built by our program is also shown in Figure 4-21. As we can see in Table 4-15 that we get accuracy of **0.99** for gini criterion.



(a). (b).

Figure 4-19. Random forest with **gini** criterion analyzed by sandbox for combined features. (a) confusion matrix (b) ROC AUC graph.

Table 4-15. Random forest with **gini** criterion model performance for combined features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.98	0.99	0.99	0.993	421
Malicious	1.00	0.99	1.00	0.993	1007
macro avg	0.99	0.99	0.99	**0.99**	1428
weighted avg	0.99	0.99	0.99	**0.99**	1428

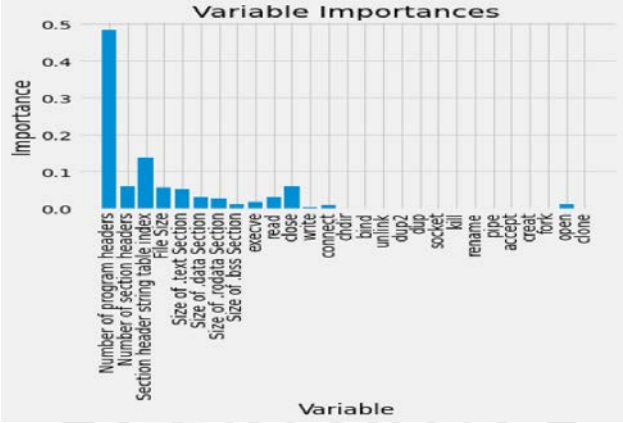


Figure 4-20. Random forest with **gini** criterion analyzed by sandbox variable importance for combined features.

Figure 4-21. Random forest with **gini** criterion analyzed by sandbox for combined features plot tree.

4.2.4 Gaussian Naïve Bayes

We examine the results of gaussian naïve bayes model trained individually on the static, system call, and combined features. For static features, our results are shown in Figure 4-22 and Table 4-16. In Figure 4-22(b), we can see that AUC=0.747. In Table 4-16,
Benign-precision=412/(412+970)=0.299, Benign-recall=412/(412+9)=0.979,
Malicious-precision=37/(9+37)=0.804, Malicious-recall=37/(970+37)=0.037,
macroavg-precision=(0.30+0.80)/2=0.55, macroavg-recall=(0.98+0.04)/2=0.51,
weightedavg-precision=(0.30*421+0.80*1007)/1428=0.66,
weightedavg-recall=(0.98*421+0.04*1007)/1428=0.31,
Benign-f1score=2*0.299*0.979/(0.299+0.979)=0.585442/1.278=0.458,
Malicious-f1score=2*0.804*0.037/(0.804+0.037)=0.059496/0.841=0.071. As we can see in Table 4-16 that we get accuracy of **0.31**.

For system call features, our results are shown in Figure 4-23 and Table 4-17. In Figure 4-23(b), we can see that AUC=0.695. In Table 4-17,
Benign-precision=418/(418+919)=0.313, Benign-recall=418/(418+3)=0.993,
Malicious-precision=88/(3+88)=0.967, Malicious-recall=88/(919+88)=0.087,
macroavg-precision=(0.31+0.97)/2=0.64, macroavg-recall=(0.99+0.09)/2=0.54,
weightedavg-precision=(0.31*421+0.97*1007)/1428=0.77,
weightedavg-recall=(0.99*421+0.09*1007)/1428=0.35,
Benign-f1score=2*0.313*0.993/(0.313+0.993)=0.621618/1.306=0.476,
Malicious-f1score=2*0.967*0.087/(0.967+0.087)=0.168258/1.054 =0.160. As we can see in Table 4-17 that we get accuracy of **0.35**.

For combined features, our results are shown in Figure 4-24 and Table 4-18. In Figure 4-24(b), we can see that AUC=0.745. In Table 4-18,

Benign-precision=417/(417+970)=0.301, Benign-recall=417/(417+4)=0.990,

Malicious-precision=37/(4+37)=0.902, Malicious-recall=37/(970+37)=0.037,

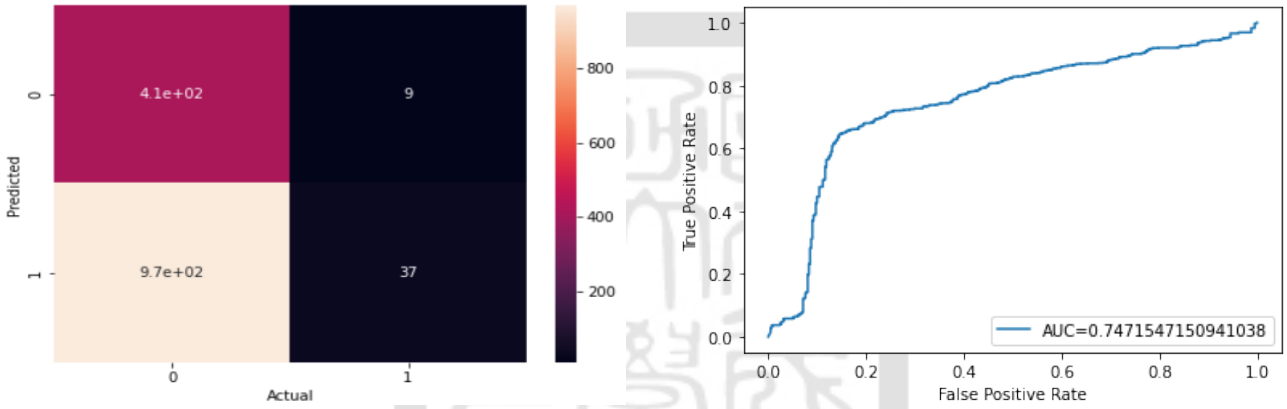
macroavg-precision=(0.30+0.90)/2=0.60, macroavg-recall=(0.99+0.04)/2=0.51,

weightedavg-precision=(0.30*421+0.90*1007)/1428=0.73,

weightedavg-recall=(0.99*421+0.04*1007)/1428=0.32,

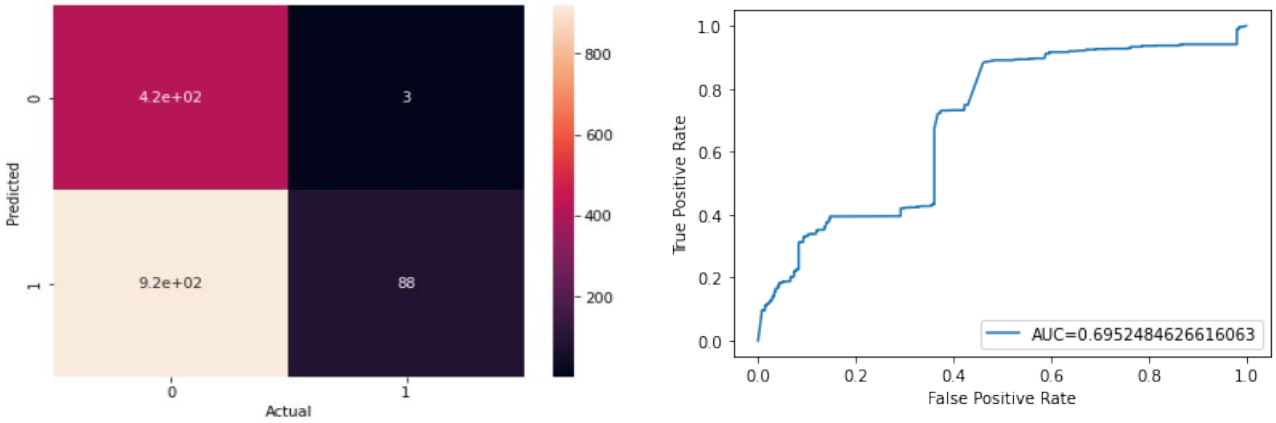
Benign-f1score=2*0.301*0.990/(0.301+0.990)=0.59598/1.291=0.462,

Malicious-f1score=2*0.901*0.037/(0.901+0.037)=0.066674/0.938 =0.071. As we can see in Table 4-18 that we get accuracy of **0.32**.



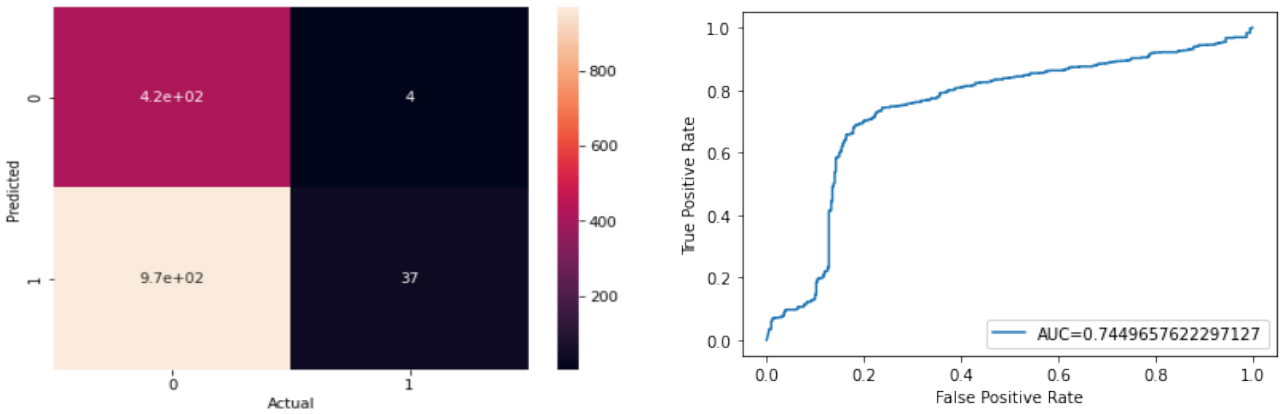
(a). (b).

Figure 4-22. Gaussian Naïve Bayes analyzed by sandbox for static features. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-23. Gaussian Naïve Bayes analyzed by sandbox for system call features. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-24. Gaussian Naïve Bayes analyzed by sandbox for combined features. (a) confusion matrix (b) ROC AUC graph.

Table 4-16. Gaussian Naïve Bayes model performance for static features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.30	0.98	0.46	0.979	421
Malicious	0.80	0.04	0.07	0.037	1007
macro avg	0.55	0.51	0.26	**0.31**	1428
weighted avg	0.66	0.31	0.18	**0.31**	1428

Table 4-17. Gaussian Naïve Bayes model performance for system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.31	0.99	0.48	0.993	421
Malicious	0.97	0.09	0.16	0.087	1007
macro avg	0.64	0.54	0.32	**0.35**	1428
weighted avg	0.77	0.35	0.25	**0.35**	1428

Table 4-18. Gaussian Naïve Bayes model performance for combined features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.30	0.99	0.46	0.990	421
Malicious	0.90	0.04	0.07	0.037	1007
macro avg	0.60	0.51	0.27	**0.32**	1428
weighted avg	0.73	0.32	0.19	**0.32**	1428

4.2.5 Support Vector Machine

We examine the results of support vector machine model with SVC function trained individually on the static, system call, and combined features. For static features, our results using SVC function are shown in Figure 4-25 and Table 4-19. In Figure 4-25(b), we can see that AUC=0.680.
In Table 4-19, Benign-precision=0/(0+0)=0.000, Benign-recall=0/(0+421)=0.000, Malicious-precision=1007/(421+1007)=0.705, Malicious-recall=1007/(0+1007)=1.000, macroavg-precision=(0.00+0.71)/2=0.35, macroavg-recall=(0.00+1.00)/2=0.50, weightedavg-precision=(0.00*421+0.71*1007)/1428=0.50, weightedavg-recall=(0.00*421+1.00*1007)/1428=0.71, Benign-f1score=2*0.000*0.000/(0.000+0.000)=0.000/0.000=0.000, Malicious-f1score=2*0.705*1.000/(0.705+1.000)=1.41/1.705=0.827. As we can see in Table 4-19 that we get accuracy of **0.71** for SVC criterion.

For system call features, our results using SVC function are shown in Figure 4-26 and Table 4-20. In Figure 4-26(b), we can see that AUC=0.759. In Table 4-20, Benign-precision=0/(0+0)=0.000, Benign-recall=0/(0+421)=0.000, Malicious-precision=1007/(421+1007)=0.705, Malicious-recall=1007/(0+1007)=1.000, macroavg-precision=(0.00+0.71)/2=0.35, macroavg-recall=(0.00+1.00)/2=0.50, weightedavg-precision=(0.00*421+0.71*1007)/1428=0.50, weightedavg-recall=(0.00*421+1.00*1007)/1428=0.71, Benign-f1score=2*0.000*0.000/(0.000+0.000)=0.000/0.000=0.000, Malicious-f1score=2*0.705*1.000/(0.705+1.000)=1.41/1.705=0.827. As we can see in Table 4-20 that we get accuracy of **0.71** for SVC criterion.

For combined features, our results using SVC function are shown in Figure 4-27 and Table 4-21. In Figure 4-27(b), we can see that AUC=0.683. In Table 4-21,

Benign-precision=0/(0+0)=0.000, Benign-recall=0/(0+421)=0.000,

Malicious-precision=1007/(421+1007)=0.705, Malicious-recall=1007/(0+1007)=1.000,

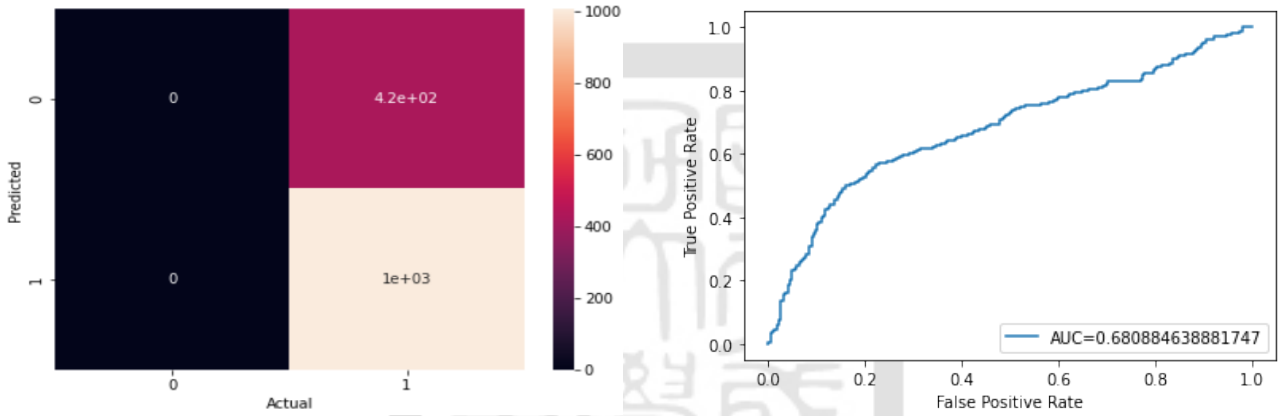
macroavg-precision=(0.00+0.71)/2=0.35, macroavg-recall=(0.00+1.00)/2=0.50,

weightedavg-precision=(0.00*421+0.71*1007)/1428=0.50,

weightedavg-recall=(0.00*421+1.00*1007)/1428=0.71,

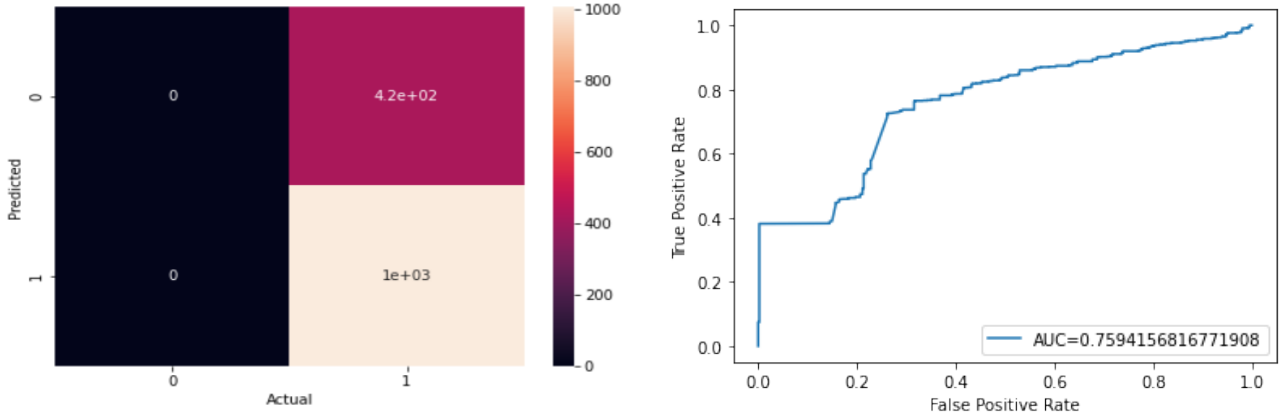
Benign-f1score=2*0.000*0.000/(0.000+0.000)=0.000/0.000=0.000,

Malicious-f1score=2*0.705*1.000/(0.705+1.000)=1.41/1.705=0.827. As we can see in Table 4-21 that we get accuracy of **0.71** for SVC criterion.



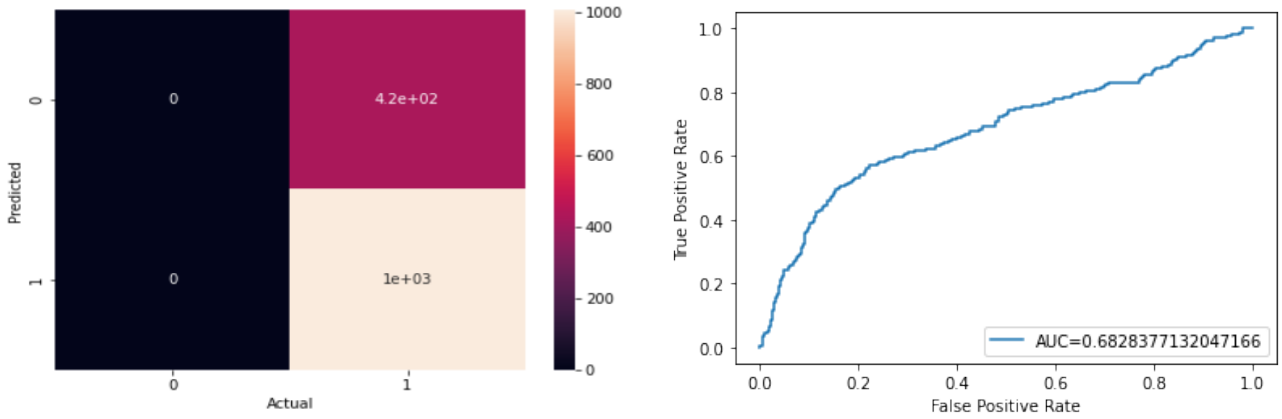
(a). (b).

Figure 4-25. SVM SVC analyzed by sandbox for static features. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-26. SVM SVC analyzed by sandbox for system call features. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-27. SVM SVC analyzed by sandbox for combined features. (a) confusion matrix (b) ROC AUC graph.

Table 4-19. SVM SVC model performance for static features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.00	0.00	0.00	0.00	421
Malicious	0.71	1.00	0.83	1.00	1007
macro avg	0.35	0.50	0.41	**0.71**	1428
weighted avg	0.50	0.71	0.58	**0.71**	1428

Table 4-20. SVM SVC model performance for system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.00	0.00	0.00	0.00	421
Malicious	0.71	1.00	0.83	1.00	1007
macro avg	0.35	0.50	0.41	**0.71**	1428
weighted avg	0.50	0.71	0.58	**0.71**	1428

Table 4-21. SVM SVC model performance for combined features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.00	0.00	0.00	0.00	421
Malicious	0.71	1.00	0.83	1.00	1007
macro avg	0.35	0.50	0.41	**0.71**	1428
weighted avg	0.50	0.71	0.58	**0.71**	1428

4.2.6 K-Nearest Neighbor

We examine the results of KNN model trained individually on the static, system call, and combined features. For static features, Figure 4-28(a) shows F1-score with different k values and Figure 4-28(b) shows error rate with different k values. Our results are shown in Figure 4-29, 4-30, 4-31, Table 4-22, 4-23 and 4-24.

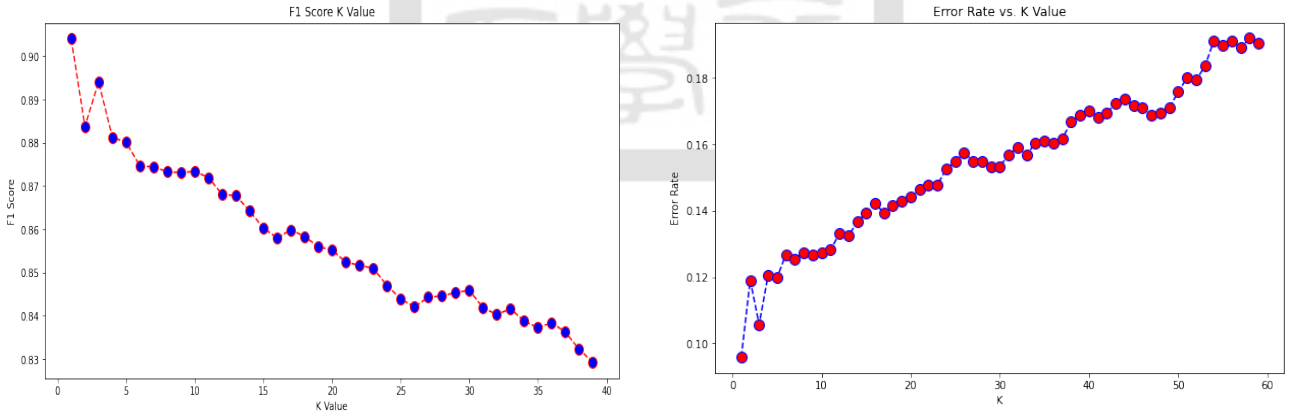
In Figure 4-29(b), we can see that AUC=0.884. In Table 4-22, Benign-precision=352/(352+68)=0.838, Benign-recall=352/(352+69)=0.836, Malicious-precision=939/(69+939)=0.932, Malicious-recall=939/(68+939)=0.932, macroavg-precision=(0.84+0.93)/2=0.88, macroavg-recall=(0.84+0.93)/2=0.88, weightedavg-precision=(0.84*421+0.93*1007)/1428=0.90, weightedavg-recall=(0.84*421+0.93*1007)/1428=0.90, Benign-f1score=2*0.838*0.836/(0.838+0.836)=1.401136/1.674=0.837, Malicious-f1score=2*0.932*0.932/(0.932+0.932)=1.737428/1.864=0.932.

In Figure 4-30(b), we can see that AUC=0.915. In Table 4-23, Benign-precision=379/(379+128)=0.748, Benign-recall=379/(379+42)=0.900, Malicious-precision=879/(42+879)=0.954, Malicious-recall=879/(128+879)=0.873, macroavg-precision=(0.75+0.95)/2=0.85, macroavg-recall=(0.90+0.87)/2=0.89, weightedavg-precision=(0.75*421+0.95*1007)/1428=0.89, weightedavg-recall=(0.90*421+0.87*1007)/1428=0.88, Benign-f1score=2*0.748*0.900/(0.748+0.900)=1.3464/1.648=0.817, Malicious-f1score=2*0.954*0.873/(0.954+0.873)=1.665684/1.827=0.912.

In Figure 4-31(b), we can see that AUC=0.930. In Table 4-24, Benign-precision=343/(343+73)=0.825, Benign-recall=343/(343+78)=0.815, Malicious-precision=934/(78+934)=0.923, Malicious-recall=934/(73+934)=0.928,

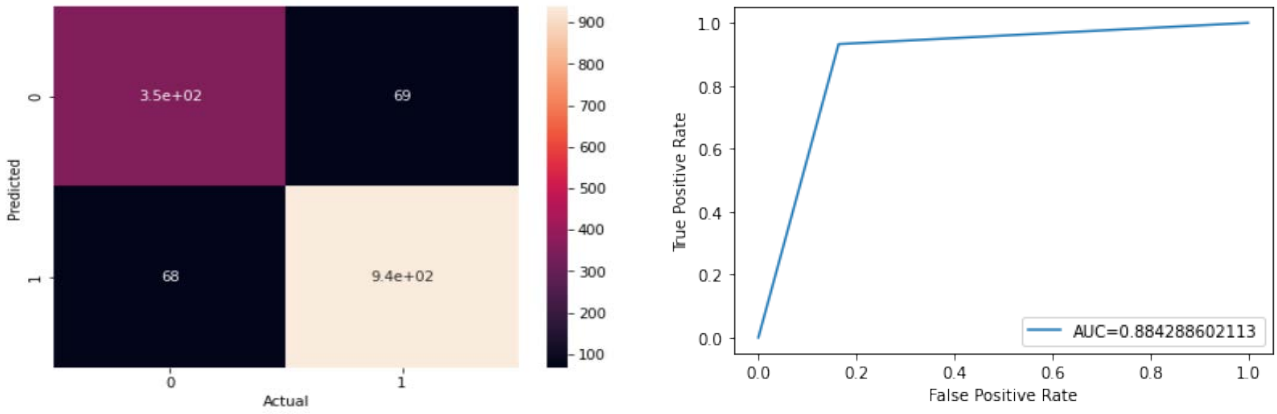
macroavg-precision=(0.82+0.92)/2=0.87, macroavg-recall=(0.81+0.93)/2=0.87, weightedavg-precision=(0.82*421+0.92*1007)/1428=0.89, weightedavg-recall=(0.81*421+0.93*1007)/1428=0.89, Benign-f1score=2*0.825*0.815/(0.825+0.815)=1.34475/1.64=0.820, Malicious-f1score=2*0.923*0.928/(0.923+0.928)=1.713088/1.851=0.925. As we can see in Table 4-22, 4-23 and 4-24 that we get accuracy when K=1, we get **0.904**; when K=2, we get **0.881**; when K=3, we get **0.894**.

It can be found in KNN ELF files have been analyzed by the sandbox static features shown at Figure 4-28(a), starting from K=4, shows that the f1-score decreases steadily. The choice of k value reflects the trade-off between bias and variance: the smaller the k value, the complex model, small bias and large variance (small training error, large test error), prone to overfitting; the larger the k value, the simpler the model , large deviation and small variance (large training error, small test error), prone to underfitting; therefore, cross-validation is generally used to select a smaller optimal k value.



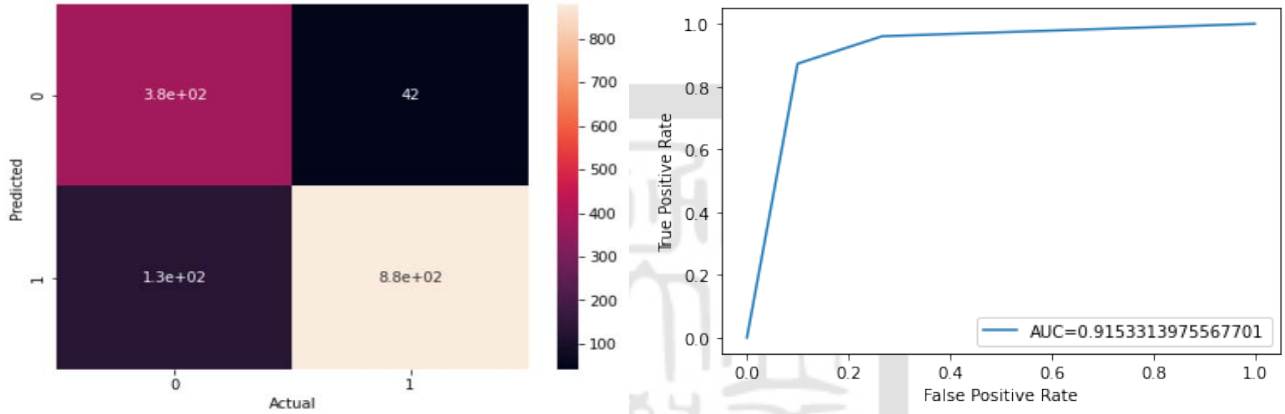
(a). (b).

Figure 4-28. KNN analyzed by sandbox for static features with different k values (a) F1-score (b) Error rate.



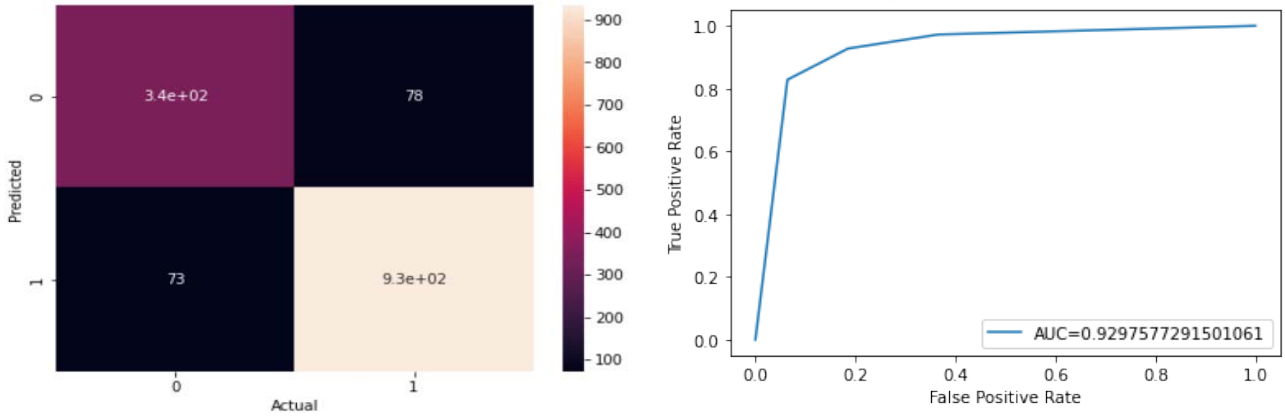
(a). (b).

Figure 4-29. KNN analyzed by sandbox for static features at k=1. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-30. KNN analyzed by sandbox for static features at k=2. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-31. KNN analyzed by sandbox for static features at k=3. (a) confusion matrix (b) ROC AUC graph.

Table 4-22. KNN model performance for static features at k=1.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.84	0.84	0.84	0.836	421
Malicious	0.93	0.93	0.93	0.932	1007
macro avg	0.88	0.88	0.88	**0.90**	1428
weighted avg	0.90	0.90	0.90	**0.90**	1428

Table 4-23. KNN model performance for static features at k=2.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.75	0.90	0.82	0.900	421
Malicious	0.95	0.87	0.91	0.873	1007
macro avg	0.85	0.89	0.86	**0.88**	1428
weighted avg	0.89	0.88	0.88	**0.88**	1428

Table 4-24. KNN model performance for static features at k=3.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.82	0.81	0.82	0.815	421
Malicious	0.92	0.93	0.93	0.928	1007
macro avg	0.87	0.87	0.87	**0.89**	1428
weighted avg	0.89	0.89	0.89	**0.89**	1428

For system call features, Figure 4-32(a) shows F1-score with different k values and Figure 4-32(b) shows error rate with different k values. Our results are shown in Figure 4-33, 4-34, 4-35, 4-36, 4-37, 4-38, 4-39, Table 4-25, 4-26, 4-27, 4-28, 4-29, 4-30 and 4-31.

In Figure 4-33(b), we can see that AUC=0.884. In Table 4-25, Benign-precision=349/(349+62)=0.849, Benign-recall=349/(349+72)=0.829, Malicious-precision=945/(72+945)=0.929, Malicious-recall=945/(62+945)=0.938, macroavg-precision=(0.85+0.93)/2=0.89, macroavg-recall=(0.83+0.94)/2=0.88, weightedavg-precision=(0.85*421+0.93*1007)/1428=0.91,

weightedavg-recall=(0.83*421+0.94*1007)/1428=0.91,

Benign-f1score=2*0.849*0.829/(0.849+0.829)=1.407642/1.678=0.839,

Malicious-f1score=2*0.929*0.938/(0.929+0.938)=1.742804/1.867=0.933.

In Figure 4-34(b), we can see that AUC=0.915. In Table 4-26, Benign-precision=375/(375+107)=0.778, Benign-recall=375/(375+46)=0.891, Malicious-precision=900/(46+900)=0.951, Malicious-recall=900/(107+900)=0.894, macroavg-precision=(0.78+0.95)/2=0.86, macroavg-recall=(0.89+0.89)/2=0.89, weightedavg-precision=(0.78*421+0.95*1007)/1428=0.90, weightedavg-recall=(0.89*421+0.89*1007)/1428=0.89, Benign-f1score=2*0.778*0.891/(0.778+0.891)=1.386396/1.669=0.831, Malicious-f1score=2*0.951*0.894/(0.951+0.894)=1.700388/1.845=0.922.

In Figure 4-35(b), we can see that AUC=0.936. In Table 4-27, Benign-precision=340/(340+60)=0.85, Benign-recall=340/(340+81)=0.808, Malicious-precision=947/(81+947)=0.921, Malicious-recall=947/(60+947)=0.940, macroavg-precision=(0.85+0.92)/2=0.89, macroavg-recall=(0.81+0.94)/2=0.87, weightedavg-precision=(0.85*421+0.92*1007)/1428=0.90, weightedavg-recall=(0.81*421+0.94*1007)/1428=0.90, Benign-f1score=2*0.85*0.808/(0.85+0.808)=1.3736/1.658=0.828, Malicious-f1score=2*0.921*0.940/(0.921+0.940)=1.73148/1.861=0.930.

In Figure 4-36(b), we can see that AUC=0.937. In Table 4-28, Benign-precision=370/(370+108)=0.774, Benign-recall=370/(370+51)=0.879, Malicious-precision=899/(51+899)=0.946, Malicious-recall=899/(108+899)=0.893, macroavg-precision=(0.77+0.95)/2=0.86, macroavg-recall=(0.88+0.89)/2=0.89, weightedavg-precision=(0.77*421+0.95*1007)/1428=0.90, weightedavg-recall=(0.88*421+0.89*1007)/1428=0.89,

Benign-f1score=2*0.774*0.879/(0.774+0.879)=1.360692/1.653=0.823,

Malicious-f1score=2*0.946*0.893/(0.946+0.893)=1.689556/1.839=0.919.

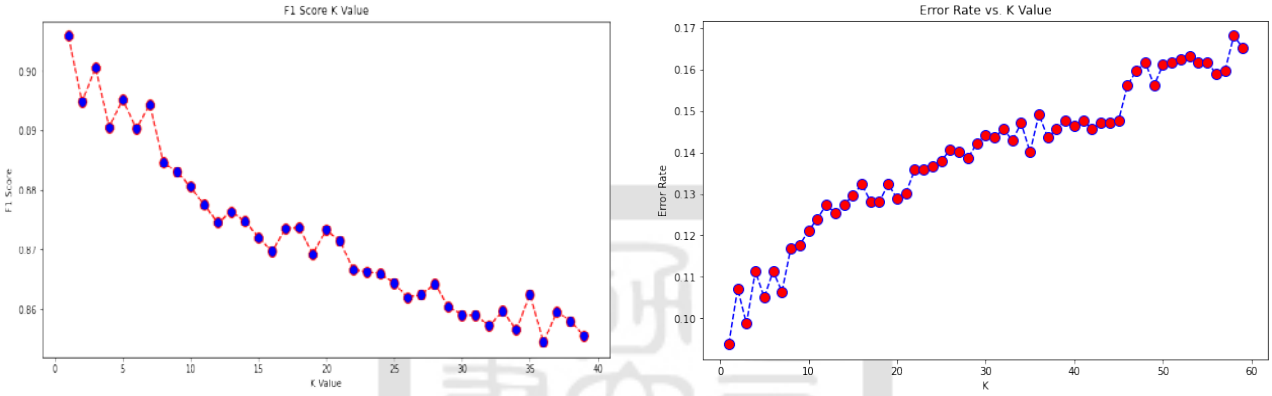
In Figure 4-37(b), we can see that AUC=0.938. In Table 4-29, Benign-precision=349/(349+78)=0.817, Benign-recall=349/(349+72)=0.829, Malicious-precision=929/(72+929)=0.928, Malicious-recall=929/(78+929)=0.923, macroavg-precision=(0.82+0.93)/2=0.87, macroavg-recall=(0.83+0.92)/2=0.88, weightedavg-precision=(0.82*421+0.93*1007)/1428=0.90, weightedavg-recall=(0.83*421+0.92*1007)/1428=0.89, Benign-f1score=2*0.817*0.829/(0.817+0.829)=1.354586/1.646=0.823, Malicious-f1score=2*0.928*0.923/(0.928+0.923)=1.713088/1.851=0.925.

In Figure 4-38(b), we can see that AUC=0.940. In Table 4-30, Benign-precision=365/(365+103)=0.780, Benign-recall=365/(365+56)=0.867, Malicious-precision=904/(56+904)=0.942, Malicious-recall=904/(103+904)=0.898, macroavg-precision=(0.78+0.94)/2=0.86, macroavg-recall=(0.87+0.90)/2=0.88, weightedavg-precision=(0.78*421+0.94*1007)/1428=0.89, weightedavg-recall=(0.87*421+0.90*1007)/1428=0.89, Benign-f1score=2*0.780*0.867/(0.780+0.867)=1.35252/1.647=0.821, Malicious-f1score=2*0.942*0.898/(0.942+0.898)=1.691832/1.84=0.919.

In Figure 4-39(b), we can see that AUC=0.941. In Table 4-31, Benign-precision=356/(356+87)=0.804, Benign-recall=356/(356+65)=0.846, Malicious-precision=920/(65+920)=0.934, Malicious-recall=920/(87+920)=0.914, macroavg-precision=(0.80+0.93)/2=0.87, macroavg-recall=(0.85+0.91)/2=0.88, weightedavg-precision=(0.80*421+0.93*1007)/1428=0.90, weightedavg-recall=(0.85*421+0.91*1007)/1428=0.89, Benign-f1score=2*0.804*0.846/(0.804+0.846)=1.360368/1.65=0.824,

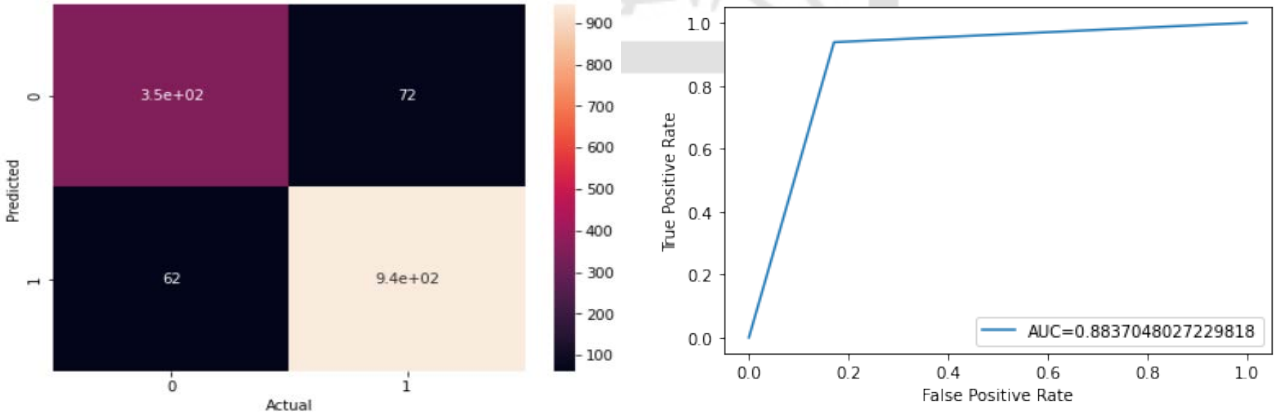
Malicious-f1score=2*0.934*0.914/(0.934+0.914)=1.707352/1.848=0.924. As we can see in Table 4-25, 4-26, 4-27, 4-28, 4-29, 4-30 and 4-31 that we get accuracy when K=1, we get **0.906**; when K=2, we get **0.893**; when K=3, we get **0.901**; when K=4, we get **0.889**; when K=5, we get **0.895**; when K=6, we get **0.889**; when K=7, we get **0.894**.

It can be found in KNN ELF files have been analyzed by the sandbox system call features shown at Figure 4-32(a), starting from K=8, the f1-score decreases steadily.



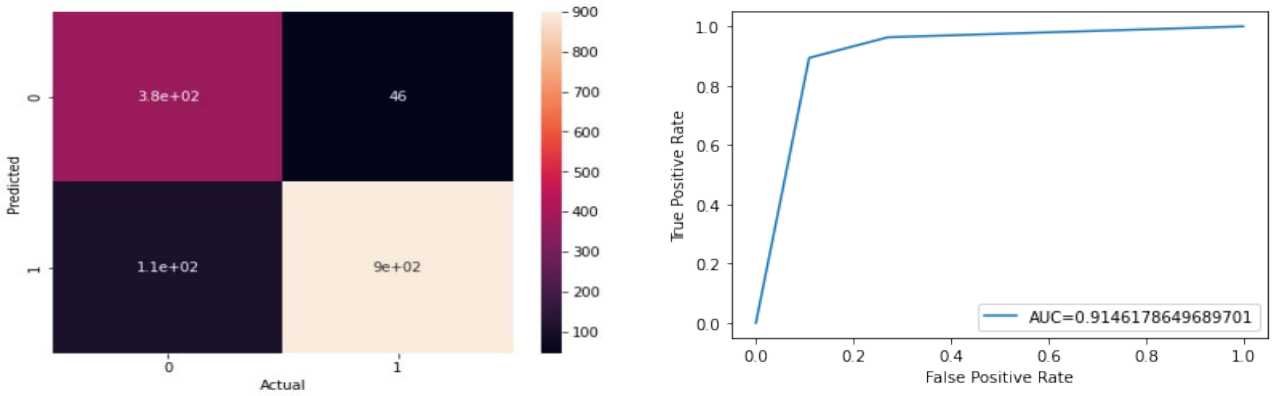
(a). (b).

Figure 4-32. KNN analyzed by sandbox for system call features with different k values (a) F1-score (b) Error rate.



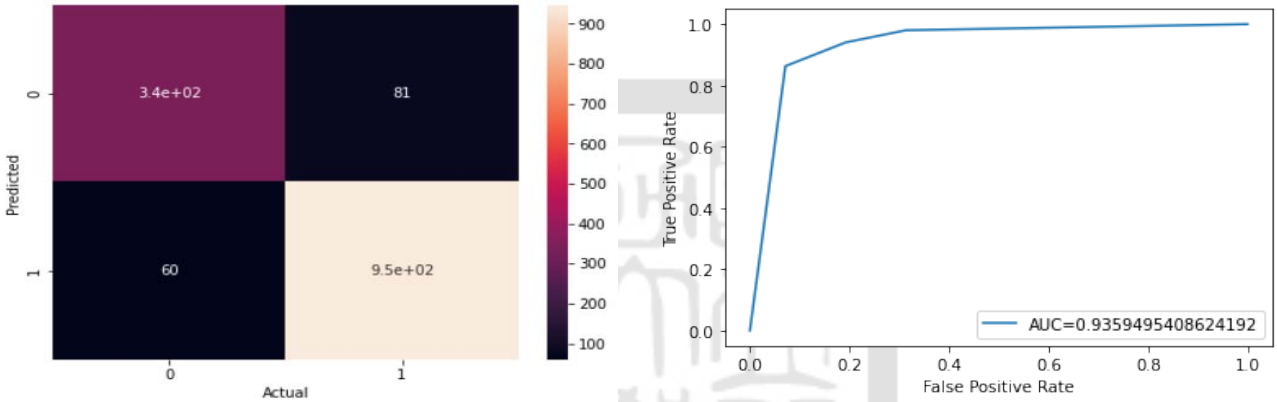
(a). (b).

Figure 4-33. KNN analyzed by sandbox for system call features at k=1. (a) confusion matrix (b) ROC AUC graph.



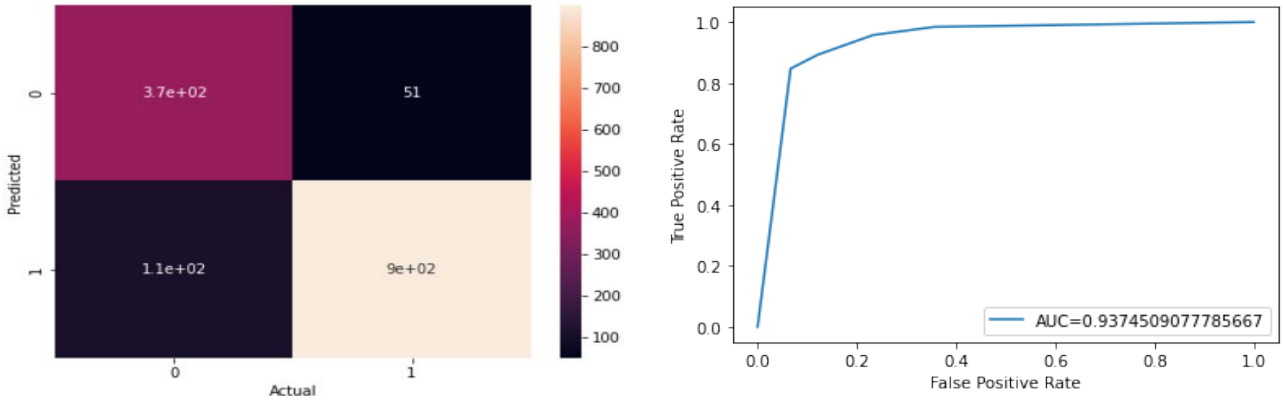
(a). (b).

Figure 4-34. KNN analyzed by sandbox for system call features at k=2. (a) confusion matrix (b) ROC AUC graph.



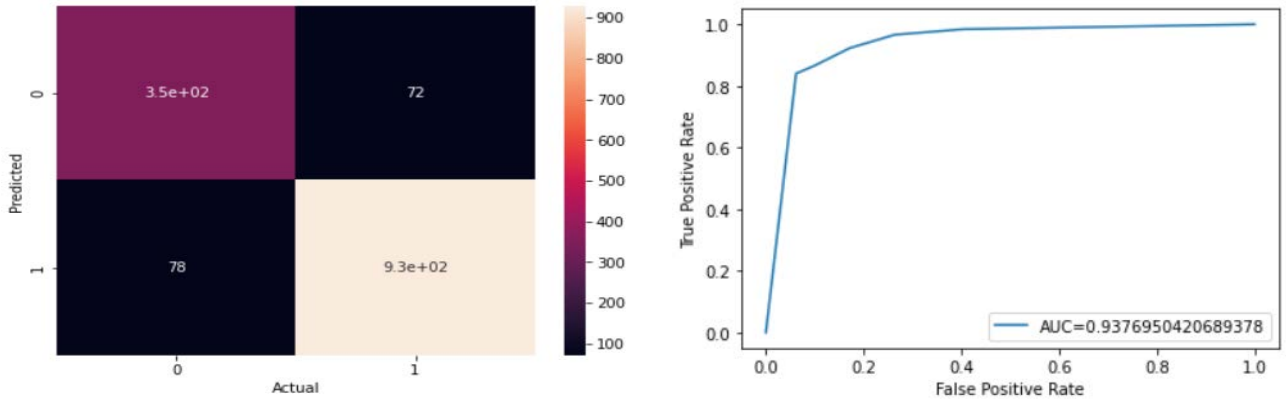
(a). (b).

Figure 4-35. KNN analyzed by sandbox for system call features at k=3. (a) confusion matrix (b) ROC AUC graph.



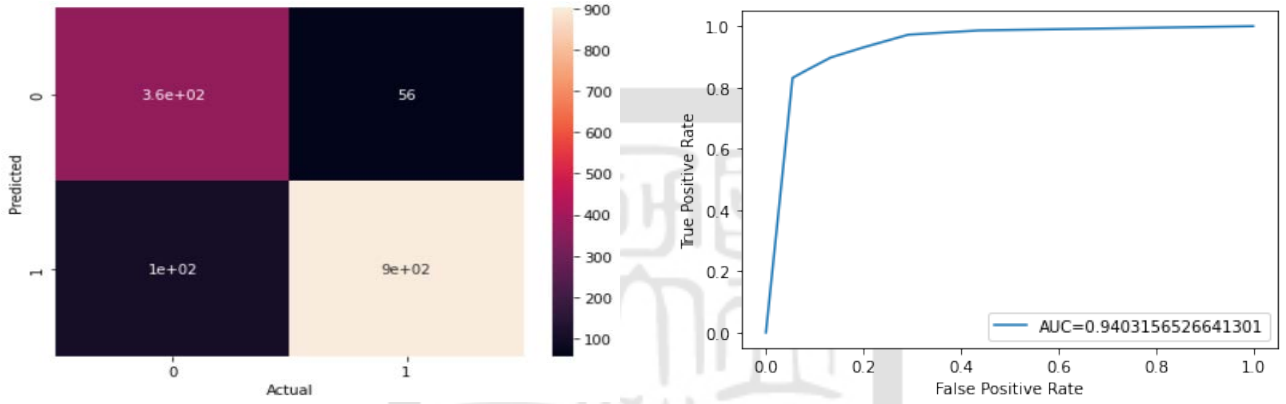
(a). (b).

Figure 4-36. KNN analyzed by sandbox for system call features at k=4. (a) confusion matrix (b) ROC AUC graph.



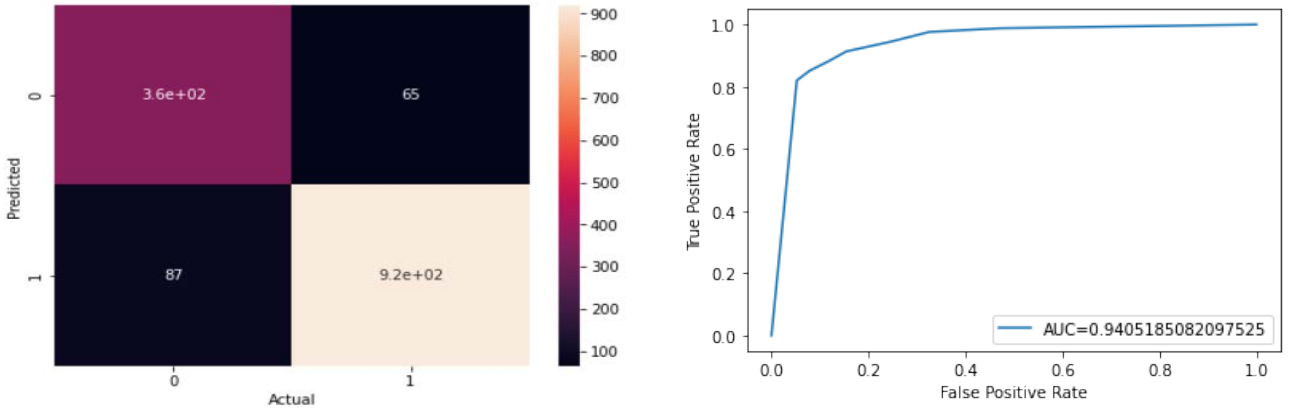
(a). (b).

Figure 4-37. KNN analyzed by sandbox for system call features at k=5. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-38. KNN analyzed by sandbox for system call features at k=6. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-39. KNN analyzed by sandbox for system call features at k=7. (a) confusion matrix (b) ROC AUC graph.

Table 4-25. KNN model performance for system call features at k=1.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.85	0.83	0.84	0.829	421
Malicious	0.93	0.94	0.93	0.938	1007
macro avg	0.89	0.88	0.89	**0.91**	1428
weighted avg	0.91	0.91	0.91	**0.91**	1428

Table 4-26. KNN model performance for system call features at k=2.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.78	0.89	0.83	0.891	421
Malicious	0.95	0.89	0.92	0.894	1007
macro avg	0.86	0.89	0.88	**0.89**	1428
weighted avg	0.90	0.89	0.89	**0.89**	1428

Table 4-27. KNN model performance for system call features at k=3.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.85	0.81	0.83	0.808	421
Malicious	0.92	0.94	0.93	0.940	1007
macro avg	0.89	0.87	0.88	**0.90**	1428
weighted avg	0.90	0.90	0.90	**0.90**	1428

Table 4-28. KNN model performance for system call features at k=4.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.77	0.88	0.82	0.879	421
Malicious	0.95	0.89	0.92	0.893	1007
macro avg	0.86	0.89	0.87	**0.89**	1428
weighted avg	0.90	0.89	0.89	**0.89**	1428

Table 4-29. KNN model performance for system call features at k=5.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.82	0.83	0.82	0.829	421
Malicious	0.93	0.92	0.93	0.923	1007
macro avg	0.87	0.88	0.87	**0.89**	1428
weighted avg	0.90	0.89	0.90	**0.89**	1428

Table 4-30. KNN model performance for system call features at k=6.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.78	0.87	0.82	0.867	421
Malicious	0.94	0.90	0.92	0.898	1007
macro avg	0.86	0.88	0.87	**0.89**	1428
weighted avg	0.89	0.89	0.89	**0.89**	1428

Table 4-31. KNN model performance for system call features at k=7.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.80	0.85	0.82	0.846	421
Malicious	0.93	0.91	0.92	0.914	1007
macro avg	0.87	0.88	0.87	**0.89**	1428
weighted avg	0.90	0.89	0.89	**0.89**	1428

For combined features, Figure 4-40(a) shows F1-score with different k values and Figure 4-40(b) shows error rate with different k values. Our results are shown in Figure 4-41, 4-42, 4-43, Table 4-32, 4-33 and 4-34. In Figure 4-41(b), we can see that AUC=0.889. In Table 4-32, Benign-precision=356/(356+68)=0.840, Benign-recall=356/(356+65)=0.846, Malicious-precision=939/(65+939)=0.935, Malicious-recall=939/(68+939)=0.932, macroavg-precision=(0.84+0.94)/2=0.89, macroavg-recall=(0.85+0.93)/2=0.89, weightedavg-precision=(0.84*421+0.94*1007)/1428=0.91, weightedavg-recall=(0.85*421+0.93*1007)/1428=0.91, Benign-f1score=2*0.840*0.846/(0.840+0.846)=1.42128/1.686=0.843, Malicious-f1score=2*0.935*0.932/(0.935+0.932)=1.74284/1.867=0.933.

In Figure 4-42(b), we can see that AUC=0.917. In Table 4-33, Benign-precision=380/(380+126)=0.751, Benign-recall=380/(380+41)=0.903, Malicious-precision=881/(41+881)=0.956, Malicious-recall=881/(126+881)=0.875, macroavg-precision=(0.75+0.96)/2=0.85, macroavg-recall=(0.90+0.87)/2=0.89,

weightedavg-precision=(0.75*421+0.96*1007)/1428=0.90,

weightedavg-recall=(0.90*421+0.87*1007)/1428=0.88,

Benign-f1score=2*0.751*0.903/(0.751+0.903)=1.356306/1.654=0.820,

Malicious-f1score=2*0.956*0.875/(0.956+0.875)=1.673/1.831=0.914.

In Figure 4-43(b), we can see that AUC=0.931. In Table 4-34,

Benign-precision=345/(345+70)=0.831, Benign-recall=345/(345+76)=0.820,

Malicious-precision=937/(76+937)=0.925, Malicious-recall=937/(70+937)=0.930,

macroavg-precision=(0.83+0.92)/2=0.88, macroavg-recall=(0.82+0.93)/2=0.87,

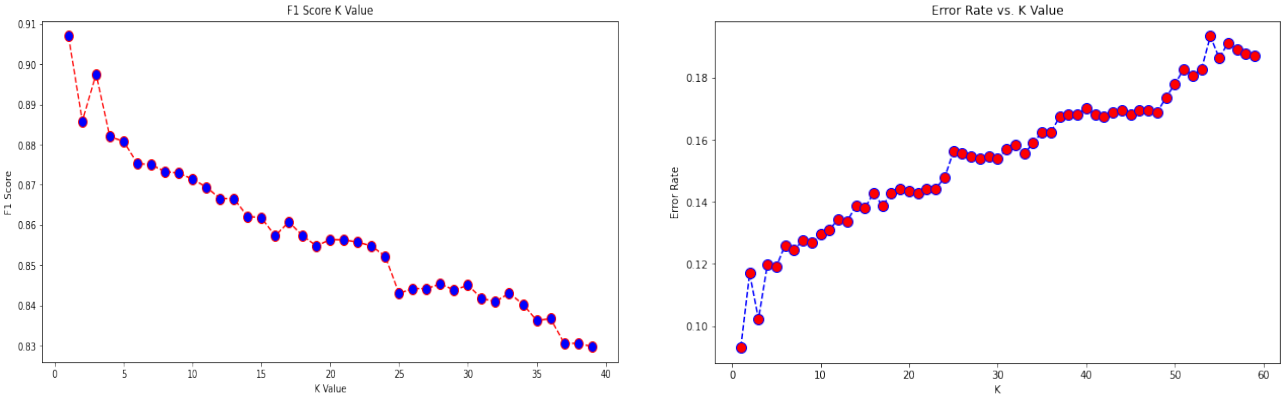
weightedavg-precision=(0.83*421+0.92*1007)/1428=0.90,

weightedavg-recall=(0.82*421+0.93*1007)/1428=0.90,

Benign-f1score=2*0.831*0.820/(0.831+0.820)=1.36284/1.651=0.825,

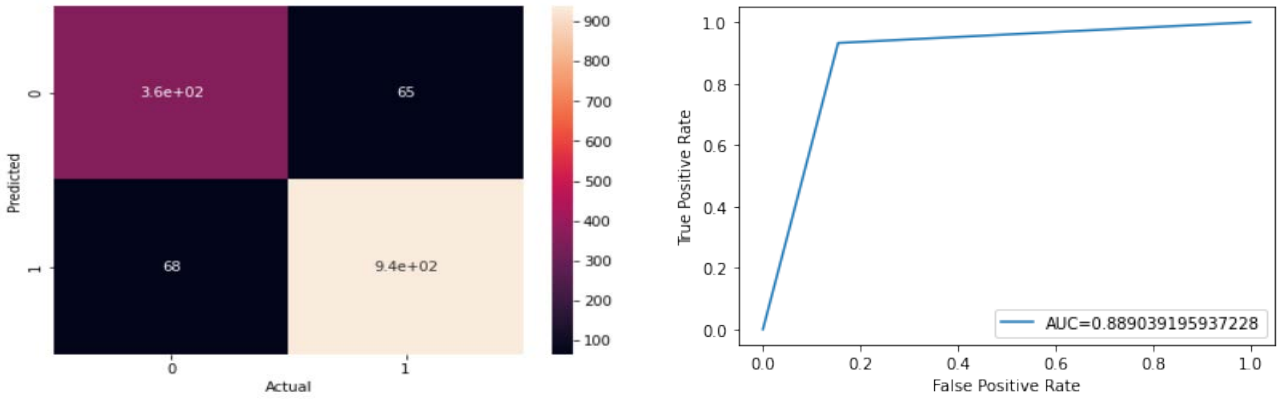
Malicious-f1score=2*0.925*0.930/(0.925+0.930)=1.7205/1.855=0.927. As we can see in Table 4-32, 4-33 and 4-34 that we get accuracy when K=1, we get **0.907**; when K=2, we get **0.883**; when K=3, we get **0.898**.

It can be found in KNN ELF files have been analyzed by the sandbox system call features shown at Figure 4-40(a), starting from K=4, the f1-score decreases steadily.



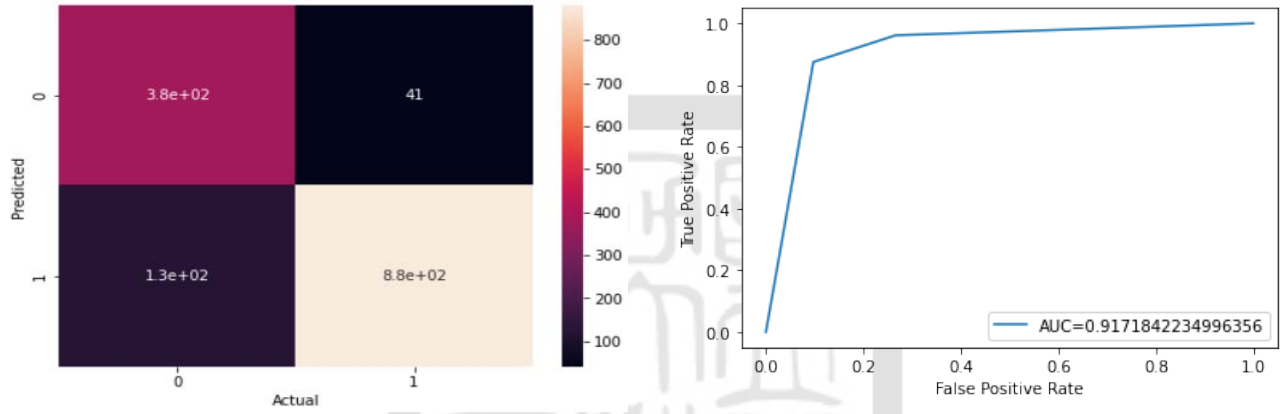
(a). (b).

Figure 4-40. KNN analyzed by sandbox for combined features with different k values (a) F1-score (b) Error rate.



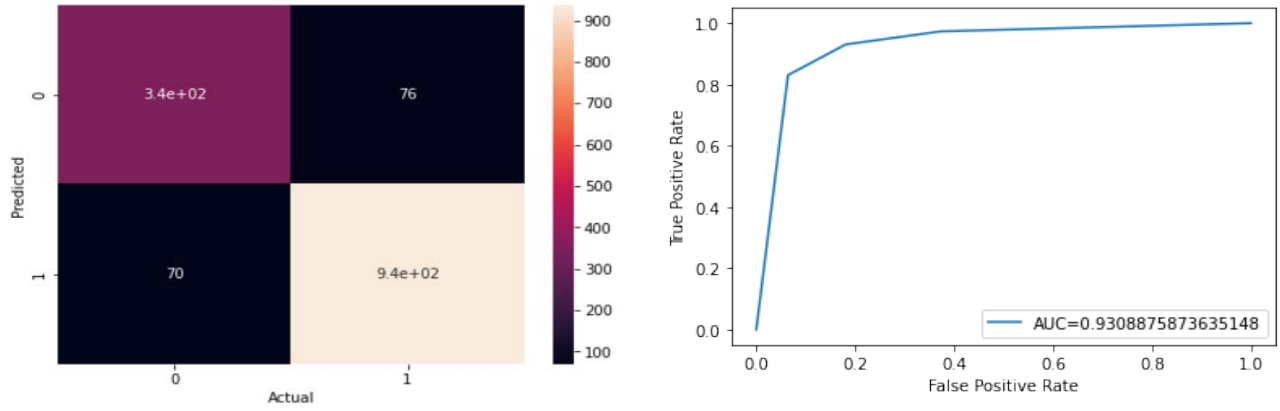
(a). (b).

Figure 4-41. KNN analyzed by sandbox for combined features at k=1. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-42. KNN analyzed by sandbox for combined features at k=2. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-43. KNN analyzed by sandbox for combined features at k=3. (a) confusion matrix (b) ROC AUC graph.

Table 4-32. KNN model performance for combined features at k=1.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.84	0.85	0.84	0.846	421
Malicious	0.94	0.93	0.93	0.932	1007
macro avg	0.89	0.89	0.89	**0.91**	1428
weighted avg	0.91	0.91	0.91	**0.91**	1428

Table 4-33. KNN model performance for combined features at k=2.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.75	0.90	0.82	0.903	421
Malicious	0.96	0.87	0.91	0.875	1007
macro avg	0.85	0.89	0.87	**0.88**	1428
weighted avg	0.90	0.88	0.89	**0.88**	1428

Table 4-34. KNN model performance for combined features at k=3.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.83	0.82	0.83	0.819	421
Malicious	0.92	0.93	0.93	0.930	1007
macro avg	0.88	0.87	0.88	**0.90**	1428
weighted avg	0.90	0.90	0.90	**0.90**	1428

In Table 4-35, we organize the accuracy obtained from the machine learning experiments in Experiments using sandbox into a table. It can be found that the accuracy of decision tree and random forest can reach above 0.9, while other machine learning models are not very effective.

Table 4-35. Machine learning model performance.

Decision tree	Static	gini	**0.99**
	System call	gini	**0.93**
	Combined	gini	**0.99**
Logistic regression	Static		**0.66**
	System call		**0.71**
	Combined		**0.67**
Random forest	Static	gini	**0.99**
	System call	gini	**0.94**
	Combined	gini	**0.99**
Gaussian naïve bayes	Static		**0.31**
	System call		**0.35**
	Combined		**0.32**
Support vector machine	Static	SVC	**0.71**
	System call	SVC	**0.71**
	Combined	SVC	**0.71**
K nearest neighbor	Static	K=1	**0.904**
		K=2	**0.881**
		K=3	**0.894**
	System call	K=1	**0.906**
		K=2	**0.893**
		K=3	**0.901**
		K=4	**0.889**
		K=5	**0.895**
		K=6	**0.889**
		K=7	**0.894**
	Combined	K=1	**0.907**
		K=2	**0.883**
		K=3	**0.898**

4.2.7 Convolutional Neural Network

In method 1, after we use **limon sandbox** to extract the feature values of each ELF file, we normalize all the feature values and convert them into 27*27 grayscale images. Figure 4-44 shows one of our ELF files converted into grayscale image, which is aa-enabled. We examine the results of convolutional neural network model when trained on combined features. Our results are shown in Figure 4-62 and Table 4-36. In Table 4-36,

Benign-precision=364/(364+0)=1.00, Benign-recall=364/(364+59)=0.861,

Malicious-precision=1005/(59+1005)=0.945, Malicious-recall=1005/(0+1005)=1.00,

macroavg-precision=(1.00+0.94)/2=0.97, macroavg-recall=(0.86+1.00)/2=0.93,

weightedavg-precision=(1.00*423+0.94*1005)/1428=0.96,

weightedavg-recall=(0.86*423+1.00*1005)/1428=0.96,

Benign-f1score=2*1.00*0.861/(1.00+0.861)=1.722/1.861=0.925,

Malicious-f1score=2*0.945*1.00/(0.945+1.00)=1.89/1.945=0.972. As we can see in Table 4-36 that we get accuracy of **0.952**.

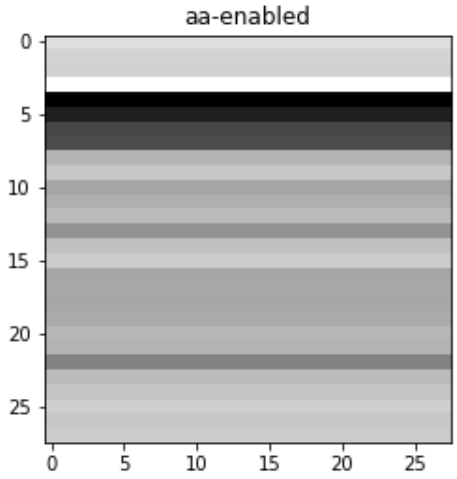


Figure 4-44. one of benign ELF files 27*27 grayscale image.

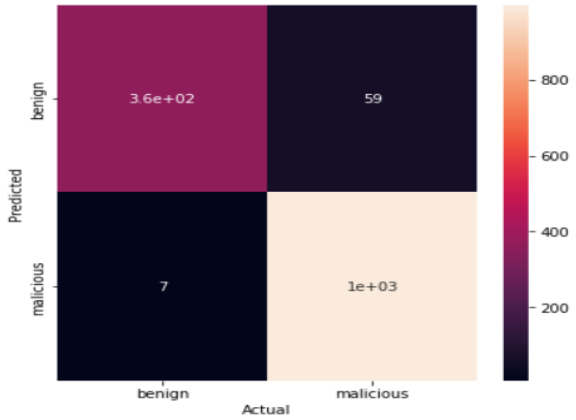
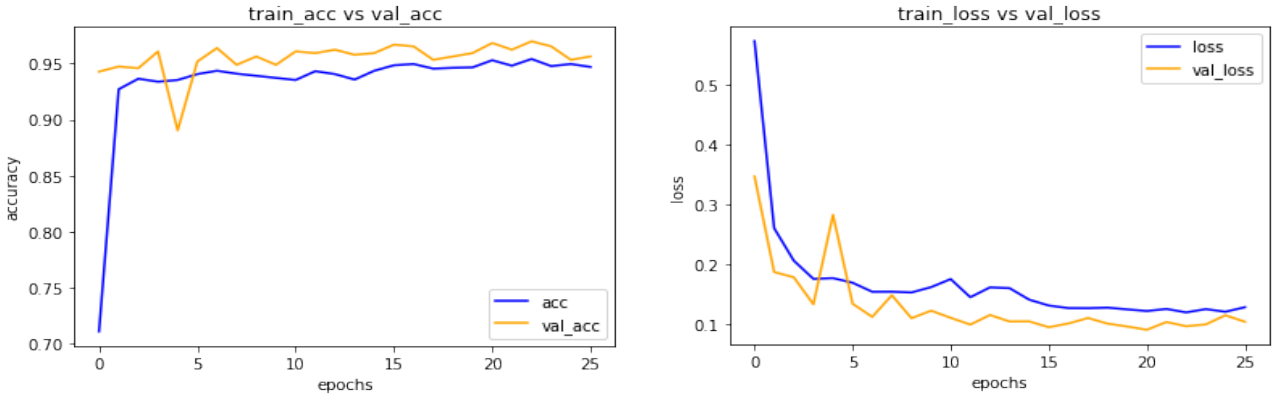


Figure 4-45. CNN analyzed by sandbox confusion matrix.

Table 4-36. CNN model performance.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	1.00	0.86	0.93	0.851	423
Malicious	0.94	1.00	0.97	0.998	1005
macro avg	0.97	0.93	0.95	**0.96**	1428
weighted avg	0.96	0.96	0.96	**0.96**	1428



(a). (b).

Figure 4-46. CNN analyzed by sandbox training and validation (a) accuracy (b) validation loss.

4.3 Experiments using strace

We used 80% of ELF files (4954*0.8 = 3963) for training and 20% of ELF files (4954 – 3963 = 991) for testing. We examine the results from Decision Tree, Logistic regression, Random Forest, Gaussian Naïve Bayes, Support Vector Machine, K-nearest neighbor based on the system call features. For all benign and malicious files, the accuracy is the percentage of the accurate predictions.

Table 4-37. Table of feature set.

Feature Type	List of Features
System Call Features	execve, brk, arch_prctl, access, mmap, close, read, pread64, mprotect, munmap, openat, fadvise64, write, exit_group

Table 4-38. System call features.

System call	description
execve	run another specified program
brk	Change the position of the program break, change the length of the data segment
arch_prctl	The function sets the specific process or thread state of the architecture
access	Checks whether the calling process can perform an operation on the specified file
mmap	map the contents of a file onto a piece of memory
close	close the file descriptor and release its corresponding kernel resources
read	reads data from an open device or file
pread64	atomically reads data from a file with an offset
mprotect	modify the protection attributes of a specified memory area
munmap	deletes the mappings for the specified address range
openat	open file relative to a directory file descriptor
fadvise64	Predeclare access modes for file data
write	writes data to an open device or file
exit_group	exit all threads in a process

	file	benign	malicious	execve	brk	arch_prctl	access	mmap	close	read	pread64	mprotect	munmap	openat	fadvise64	write	exit_group
0	aa-enabled.txt	0	1	3	2	1	2	6	3	6	3	2	1	1	1	1
1	aa-exec.txt	0	1	3	2	1	2	6	3	6	3	2	1	1	1	1
2	aconnect.txt	0	1	3	2	1	2	6	3	6	3	2	1	1	1	1
3	acpi_listen.txt	0	1	3	2	1	2	6	3	6	3	2	1	1	1	1
4	addpart.txt	0	1	3	2	1	2	6	3	6	3	2	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4949	VirusShare_ff4dbe26278bfda759ee8b1f10d94d3b.txt	1	1	3	2	1	2	6	3	6	3	2	1	1	1	1
4950	VirusShare_ff8dd015bb2766352af051944cb9781f.txt	1	1	3	2	1	2	6	3	6	3	2	1	1	1	1
4951	VirusShare_ffb00447d40b0ae015752dd484d09de8.txt	1	1	3	2	1	2	6	3	6	3	2	1	1	1	1
4952	VirusShare_ffb437621f3249c647c88350e068fd07.txt	1	1	3	2	1	2	6	3	6	3	2	1	1	1	1
4953	VirusShare_ffc7be26912b5aca63e55dc7c830f28a.txt	1	1	3	2	1	2	6	3	6	3	2	1	1	1	1

4954 rows × 16 columns

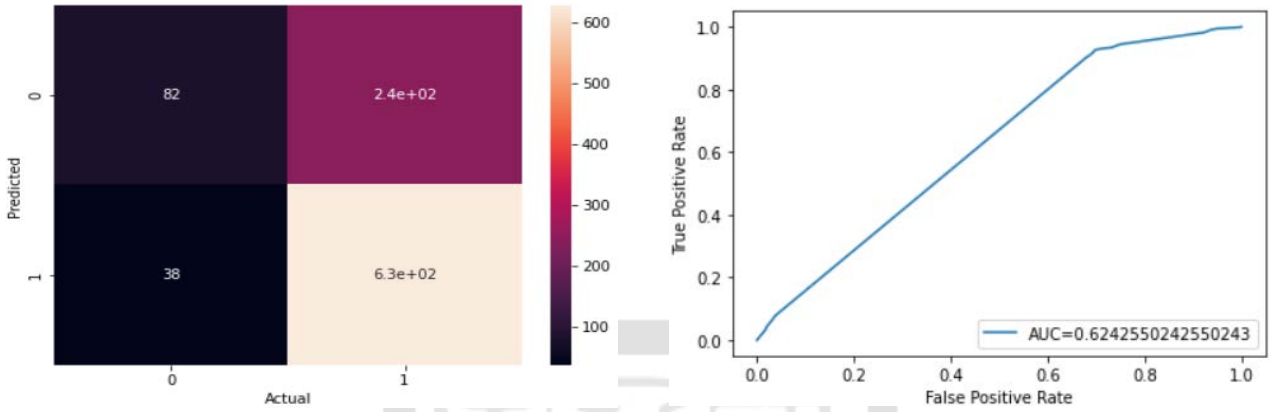
Figure 4-47. System call features dataset analyzed statically.

4.3.1 Decision Tree

We examine the results of decision tree model with gini criterion trained on the system call features. For system call features, our results using gini criterion are shown in Figure 4-48 and Table 4-40. In Figure 4-48(b), we can see that AUC=0.624. In Table 4-40, Benign-precision=82/(82+38)=0.683, Benign-recall=82/(82+243)=0.252, Malicious-precision=628/(243+628)=0.721, Malicious-recall=628/(38+628)=0.943, macroavg-precision=(0.68+0.72)/2=0.70, macroavg-recall=(0.25+0.94)/2=0.60, weightedavg-precision=(0.68*325+0.72*666)/1428=0.71, weightedavg-recall=(0.25*325+0.94*666)/1428=0.72, Benign-f1score=2*0.683*0.252/(0.683+0.252)=0.344232/0.935=0.368, Malicious-f1score=2*0.721*0.943/(0.721+0.943)=1.359806/1.664=0.817. The detailed decision trees built by our programs is also shown in Figure 4-49. From Figure 4-49, it can be found that there is still no way to completely distinguish between benign files and malicious files in the value = [benign, malicious] in the red box of the leaf node, so the accuracy can only reach 0.72. Take value= [1, 5] for example, there are one benign file and five malicious files that cannot be distinguished, qemu-img is the one benign ELF file and 363b925df467d33f6d10c6c5d70e895933b94f016b5b91f678481517e0109d0f, 4289ba60e1852a9d5ef1b4fb04be8e918148f7b3fe04fe371371f09a36efee00, 5e84a3573806c2799e9e727327235fc1db39aa62c8be0dccb1abcd206d944bb2, c36050a2036ff0a1b42f9beb667914046a16071dfb6675424a56ea3874c686ff, cbf4c57000539a44be789b19822c500ffabb6684701d55eec594608cdfe26d97. Table 4-39 shows all the feature values of these six ELF files. As we can see in Table 4-40 that we get accuracy of **0.72** for gini criterion.

Table 4-39. Decision tree with **gini** criterion for system call features value= [1, 5] feature values.

file	benign ma	execve	brk	arch_prctl	access	mmap	close	read	pread64	mprotect	munmap	openat	fadvise64	write	exit_group
qemu-img	0	1	3	2	1	2	6	18	6	3	2	1	1	16	1
363b925d	1	1	3	2	1	2	6	18	6	3	2	1	1	16	1
4289ba60	1	1	3	2	1	2	6	18	6	3	2	1	1	16	1
5e84a357	1	1	3	2	1	2	6	18	6	3	2	1	1	16	1
c36050a2	1	1	3	2	1	2	6	18	6	3	2	1	1	16	1
cbf4c5700	1	1	3	2	1	2	6	18	6	3	2	1	1	16	1



(a). (b).

Figure 4-48. Decision tree with **gini** criterion analyzed statically for system call features. (a) confusion matrix (b) ROC AUC graph.

Table 4-40. Decision tree model performance analyzed statically with **gini** criterion system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.68	0.25	0.37	0.252	325
Malicious	0.72	0.94	0.82	0.943	666
macro avg	0.70	0.60	0.59	**0.72**	991
weighted avg	0.71	0.72	0.67	**0.72**	991

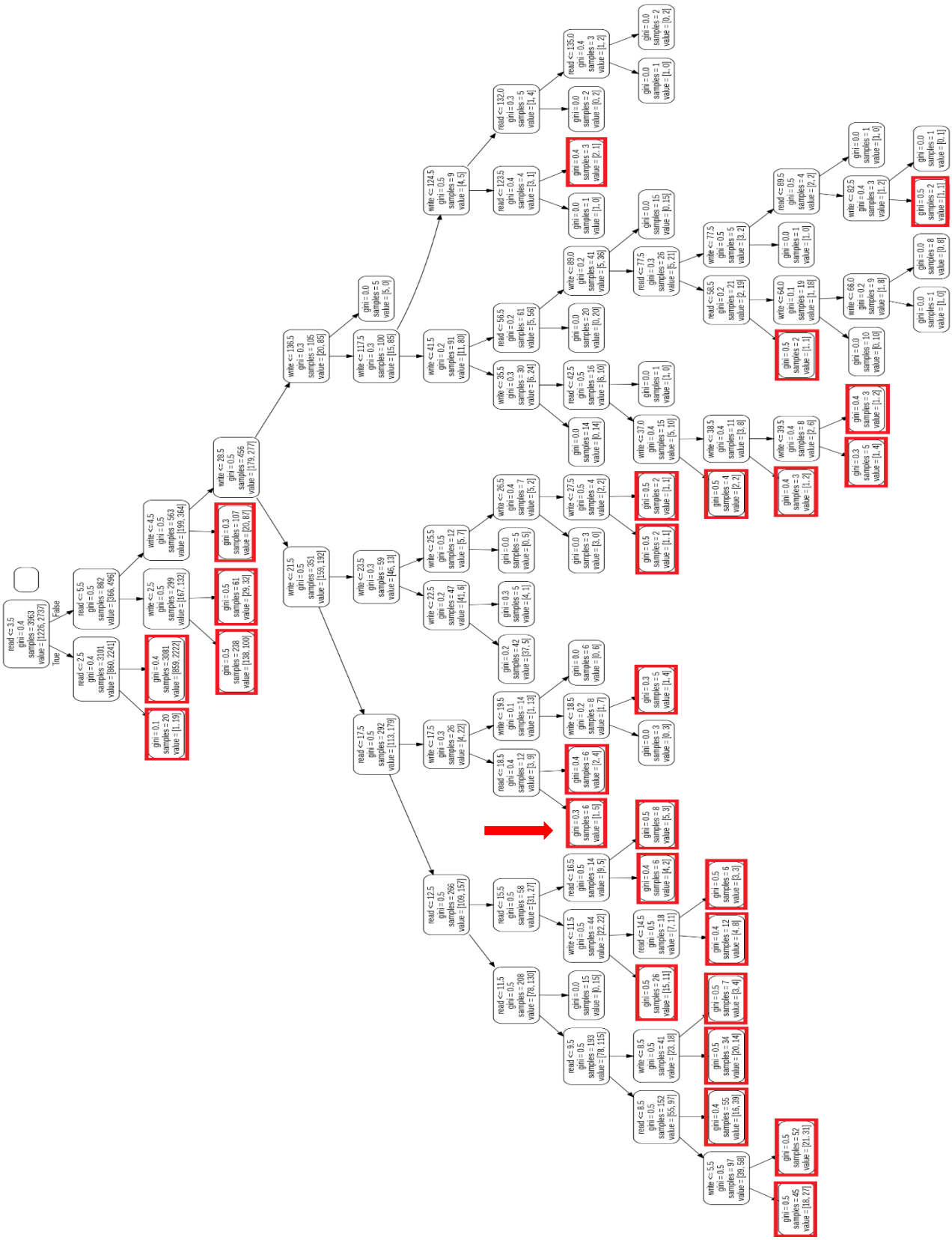


Figure 4-49. Decision tree with **gini** criterion analyzed statically for system call features plot tree.

4.3.2 Logistic Regression

We examine the results of logistic regression model trained on the system call features. For system call features, our results are shown in Figure 4-68, and Table 4-41. In Figure 4-50(b), we can see that AUC=0.582. In Table 4-41,

Benign-precision=1/(1+0)=1.00, Benign-recall=1/(1+458)=0.002,

Malicious-precision=1028/(458+1028)=0.692, Malicious-recall=1028/(0+1028)=1.00,

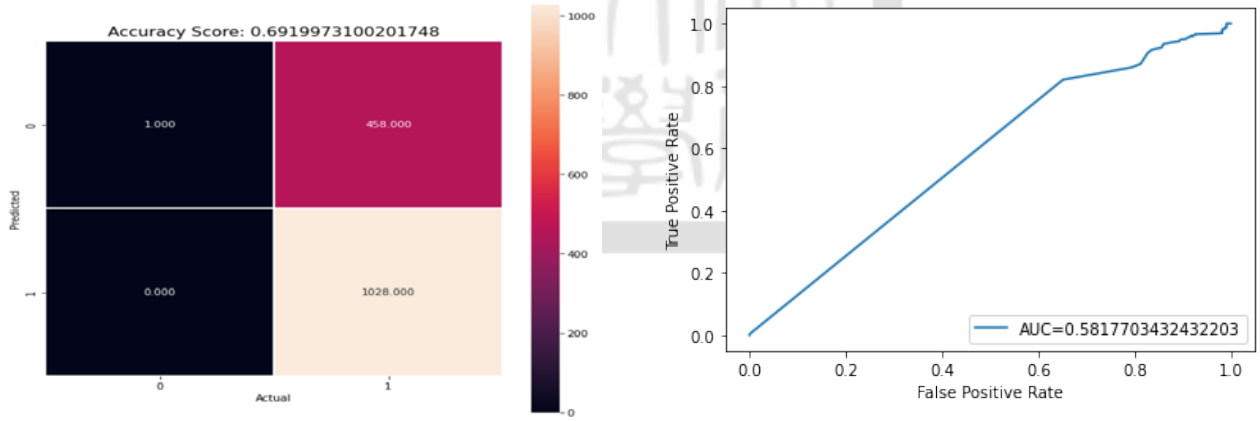
macroavg-precision=(1.00+0.69)/2=0.85, macroavg-recall=(0.00+1.00)/2=0.50,

weightedavg-precision=(1.00*459+0.69*1028)/1487=0.79,

weightedavg-recall=(1.00*459+0.69*1028)/1487=0.69,

Benign-f1score=2*1.00*0.002/(1.00+0.002)=0.004/1.002=0.004,

Malicious-f1score=2*0.692*1.00/(0.692+1.00)=1.384/1.692=0.818. As we can see in Table 4-41 that we get accuracy of **0.69**.



(a). (b).

Figure 4-50. Logistic regression analyzed statically for system call features. (a) confusion matrix (b) ROC AUC graph.

Table 4-41. Logistic regression model performance analyzed statically system call features.

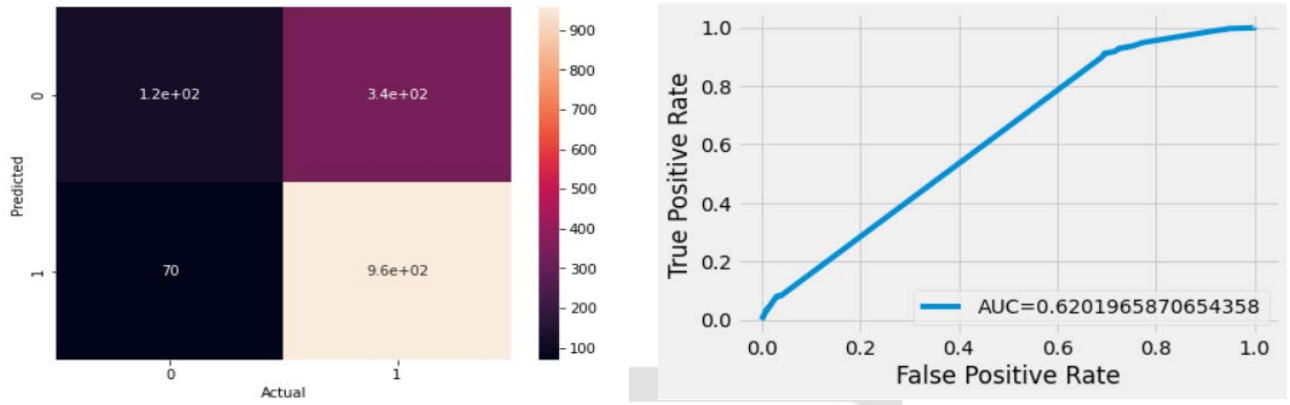
	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	1.00	0.00	0.00	0.002	459
Malicious	0.69	1.00	0.82	1.000	1028
macro avg	0.85	0.50	0.41	**0.69**	1487
weighted avg	0.79	0.69	0.57	**0.69**	1487

4.3.3 Random Forest

We examine the results of random forest model with gini criterion trained on the system call features. For system call features, our results using gini criterion are shown in Figure 4-51 and Table 4-43. In Figure 4-51(b), we can see that AUC=0.620. In Table 4-43, Benign-precision=120/(120+70)=0.632, Benign-recall=120/(120+339)=0.261, Malicious-precision=958/(339+958)=0.739, Malicious-recall=958/(70+958)=0.932, macroavg-precision=(0.63+0.74)/2=0.69, macroavg-recall=(0.26+0.93)/2=0.60, weightedavg-precision=(0.63*459+0.74*1028)/1487=0.71, weightedavg-recall=(0.26*459+0.93*1028)/1487=0.72, Benign-f1score=2*0.632*0.261/(0.632+0.261)=0.329904/0.893=0.369, Malicious-f1score=2*0.739*0.932/(0.739+0.932)=1.377496/1.671=0.824. The variable importance is shown in Figure 4-52, we found that feature “read” is the most important variable, so we choose feature “read” as our tree’s root. We set *n_estimators* to 10, which means that there are 10 trees in this forest, and we selected the 5th tree with “read” as the root. The detailed 5th of the trees in the random forest built by our program is also shown in Figure 4-53. From Figure 4-53, it can be found that there is still no way to completely distinguish between benign files and malicious files in the value = [benign, malicious] in the red box of the leaf node, so the accuracy can only reach **0.72**. Take value= [1, 6] for example, there are one benign file and five malicious files that cannot be distinguished, docker-proxy is the one benign ELF file and 03bb1cfd9e45844701aabc549f530d56f162150494b629ca19d83c1c696710d7, 1384790107a5f200cab9593a39d1c80136762b58d22d9b3f081c91d99e5d0376, 1f85b0c47432ab5abe651b8d0c0697f41b392eab9a0a966c41a623ea80432e74, 482ca6cef1615277825a821ff50a52c37ba6b4fa2f627b6fc17daea9456c8fd0, VirusShare_3aad30955ed6946670efea0dac9ba9b4,

Table 4-42. Random forest with **gini** criterion for system call features value= [1, 6] feature values.

file	benign ma	execve	brk	arch_prctl	access	mmap	close	read	pread64	mprotect	munmap	openat	fadvise64	write	exit_group
docker-proxy.txt	0	1	3	2	1	2	6	21	6	3	2	1	1	19	1
03bb1cfd9e45844701aab	1	1	3	2	1	2	6	21	6	3	2	1	1	19	1
1384790107a5f200cab95	1	1	3	2	1	2	6	21	6	3	2	1	1	19	1
1f85b0c47432ab5abe651	1	1	3	2	1	2	6	21	6	3	2	1	1	19	1
482ca6cef1615277825a8	1	1	3	2	1	2	6	21	6	3	2	1	1	19	1
VirusShare_3aad30955ec	1	1	3	2	1	2	6	21	6	3	2	1	1	19	1
a544cd35f17b851deb114	1	1	3	2	1	2	6	22	6	3	2	1	1	20	1



(a). (b).

Figure 4-51. Random forest with **gini** criterion analyzed statically for system call features. (a) confusion matrix (b) ROC AUC graph.

Table 4-43. Random forest model performance analyzed statically with **gini** criterion system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.63	0.26	0.37	0.261	459
Malicious	0.74	0.93	0.82	0.932	1028
macro avg	0.69	0.60	0.60	**0.72**	1487
weighted avg	0.71	0.72	0.68	**0.72**	1487

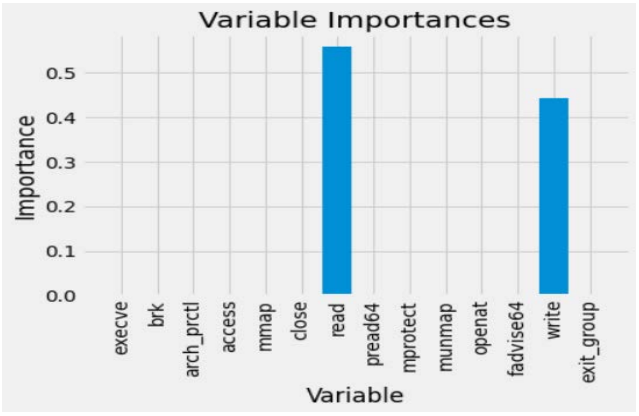


Figure 4-52. Random forest with **gini** criterion analyzed statically variable importance for system call features.

a544cd35f17b851deb1148be555351ad2c642bf8b046466094b16b37057d09d1. Table 4-42 shows all the feature values of these seven ELF files. In Table 4-43, we get accuracy of **0.72** for gini criterion.

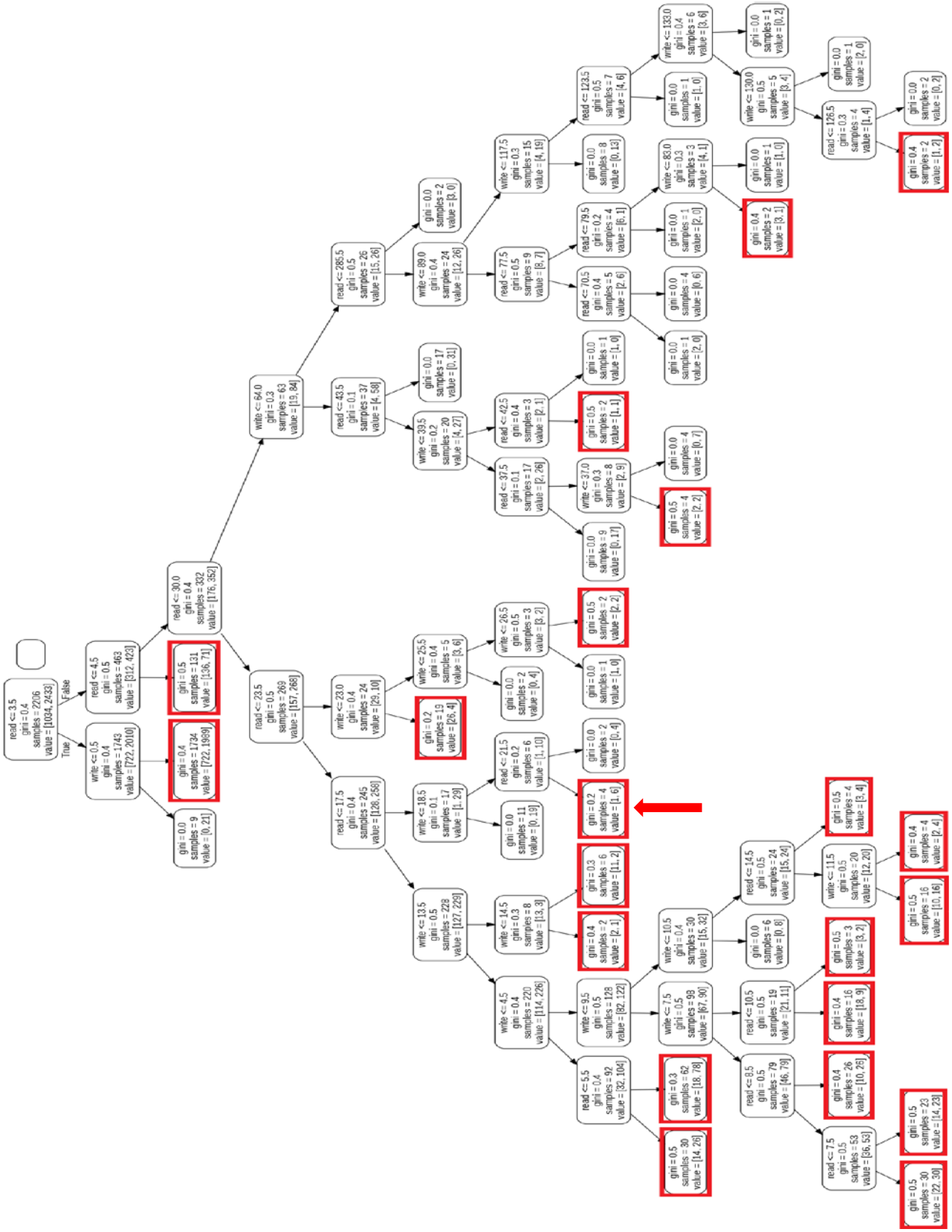


Figure 4-53. Random forest with **gini** criterion analyzed statically for system call features plot tree.

4.3.4 Gaussian Naïve Bayes

We examine the results of gaussian naïve bayes model trained on the system call features. For system call features, our results are shown in Figure 4-54 and Table 4-44. In Figure 4-54(b), we can see that AUC=0.495. In Table 4-44,

Benign-precision=9/(9+31)=0.225, Benign-recall=9/(9+450)=0.020,

Malicious-precision=997/(450+997)=0.689, Malicious-recall=997/(31+997)=0.970,

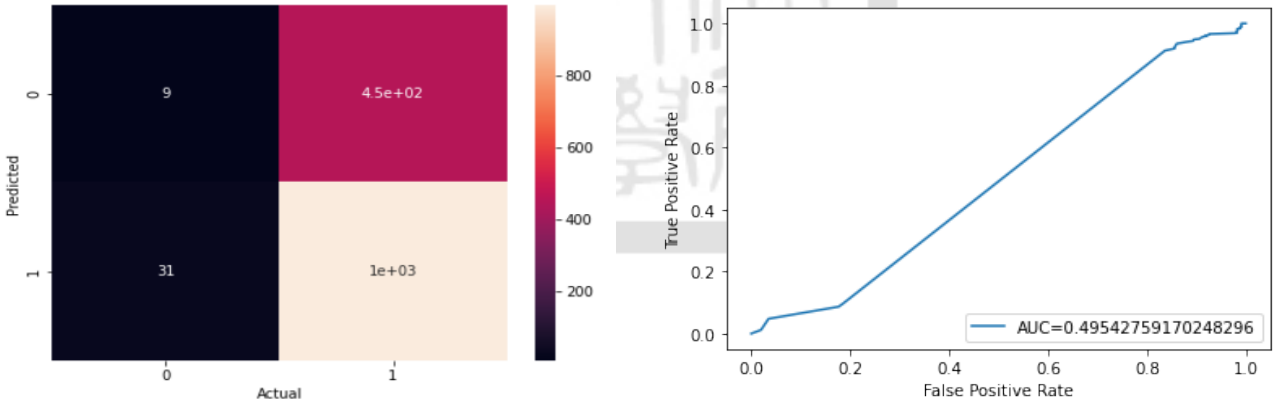
macroavg-precision=(0.23+0.69)/2=0.46, macroavg-recall=(0.02+0.97)/2=0.49,

weightedavg-precision=(0.23*459+0.69*1028)/1487=0.55,

weightedavg-recall=(0.02*459+0.97*1028)/1487=0.68,

Benign-f1score=2*0.225*0.020/(0.225+0.020)=0.009/0.245=0.037,

Malicious-f1score=2*0.689*0.970/(0.689+0.970)=1.33666/1.659=0.806. As we can see in Table 4-44 that we get accuracy of **0.68**.



(a). (b).

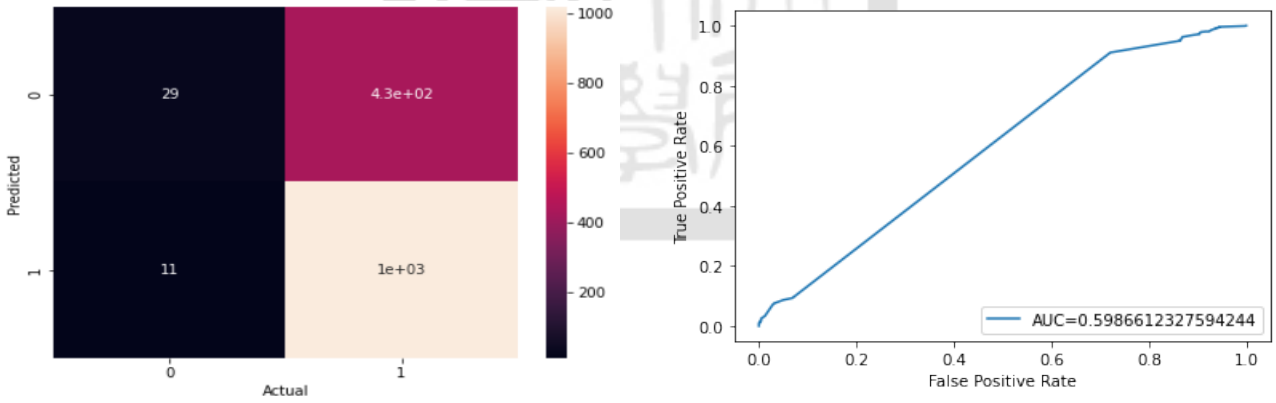
Figure 4-54. Gaussian Naïve Bayes analyzed statically for system call features. (a) confusion matrix (b) ROC AUC graph.

Table 4-44. Gaussian naïve bayes model performance analyzed statically system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.23	0.02	0.04	0.020	459
Malicious	0.69	0.97	0.81	0.970	1028
macro avg	0.46	0.49	0.42	**0.68**	1487
weighted avg	0.55	0.68	0.57	**0.68**	1487

4.3.5 Support Vector Machine

We examine the results of support vector machine model with SVC function trained on the system call features. For system call features, our results using SVC function are shown in Figure 4-58 and Table 4-45. In Figure 4-55(b), we can see that AUC=0.599. In Table 4-45, Benign-precision=29/(29+11)=0.725, Benign-recall=29/(29+430)=0.063, Malicious-precision=1017/(430+1017)=0.703, Malicious-recall=1017/(11+1017)=0.989, macroavg-precision=(0.72+0.70)/2=0.71, macroavg-recall=(0.06+0.99)/2=0.53, weightedavg-precision=(0.72*459+0.70*1028)/1428=0.71, weightedavg-recall=(0.06*459+0.99*1028)/1428=0.70, Benign-f1score=2*0.725*0.063/(0.725+0.063)=0.09135/0.788=0.116, Malicious-f1score=2*0.703*0.989/(0.703+0.989)=1.390534/1.692=0.822. As we can see in Table 4-45 that we get accuracy of **0.70** for SVC function.



(a). (b).

Figure 4-55. SVM SVC analyzed statically for system call features. (a) confusion matrix (b) ROC AUC graph.

Table 4-45. SVM SVC model performance analyzed statically system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.72	0.06	0.12	0.063	459
Malicious	0.70	0.99	0.82	0.989	1028
macro avg	0.71	0.53	0.47	**0.70**	1487
weighted avg	0.71	0.70	0.60	**0.70**	1487

4.3.6 K-Nearest Neighbor

We examine the results of KNN model trained on the system call. For system call features, Figure 4-56(a) shows F1-score with different k values and Figure 4-56(b) shows error rate with different k values. Our results are shown in Figure 4-57, 4-58, 4-59, Table 4-46, 4-47 and 4-48.

In Figure 4-57(b), we can see that AUC=0.535. In Table 4-46, Benign-precision=41/(41+19)=0.683, Benign-recall=41/(41+418)=0.089, Malicious-precision=1009/(418+1009)=0.707, Malicious-recall=1009/(19+1009)=0.982, macroavg-precision=(0.68+0.71)/2=0.70, macroavg-recall=(0.09+0.98)/2=0.54, weightedavg-precision=(0.68*459+0.71*1028)/1487=0.70, weightedavg-recall=(0.09*459+0.98*1028)/1487=0.71, Benign-f1score=2*0.683*0.089/(0.683+0.089)=0.121574/0.772=0.157, Malicious-f1score=2*0.707*0.982/(0.707+0.982)=1.388548/1.689=0.822.

In Figure 4-58(b), we can see that AUC=0.546. In Table 4-47, Benign-precision=426/(426+919)=0.317, Benign-recall=426/(426+33)=0.928, Malicious-precision=109/(33+109)=0.768, Malicious-recall=109/(919+109)=0.106, macroavg-precision=(0.32+0.77)/2=0.54, macroavg-recall=(0.93+011)/2=0.52, weightedavg-precision=(0.32*459+0.77*1028)/1487=0.63, weightedavg-recall=(0.09*459+0.98*1028)/1428=0.36, Benign-f1score=2*0.317*0.928/(0.317+0.928)=0.588352/1.245=0.473, Malicious-f1score=2*0.768*0.106/(0.768+0.106)=0.162816/0.874=0.186.

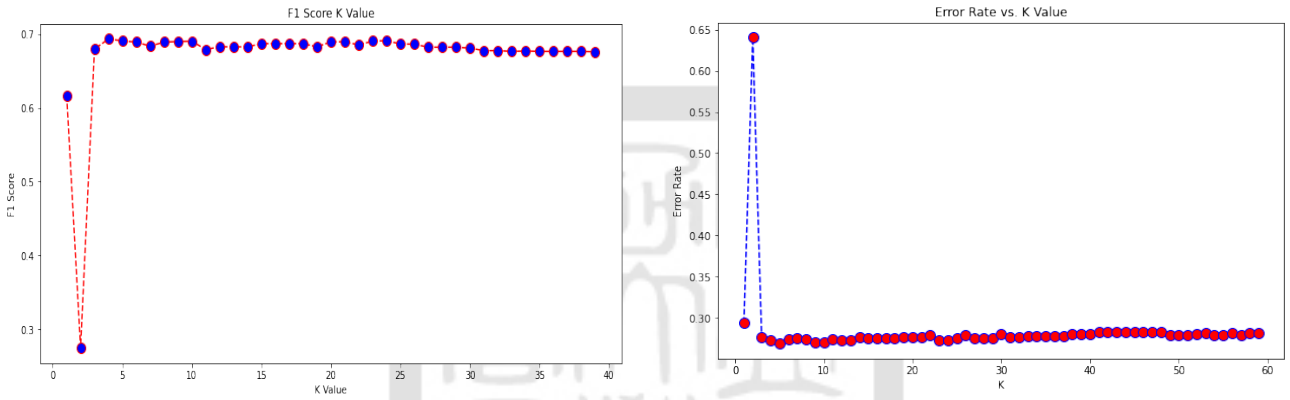
In Figure 4-59(b), we can see that AUC=0.613. In Table 4-48, Benign-precision=114/(114+66)=0.633, Benign-recall=114/(114+345)=0.248, Malicious-precision=962/(345+962)=0.736, Malicious-recall=962/(66+962)=0.936, macroavg-precision=(0.63+0.74)/2=0.68, macroavg-recall=(0.25+0.94)/2=0.59,

weightedavg-precision=(0.63*459+0.74*1028)/1487=0.70,

weightedavg-recall=(0.25*459+0.94*1028)/1428=0.72,

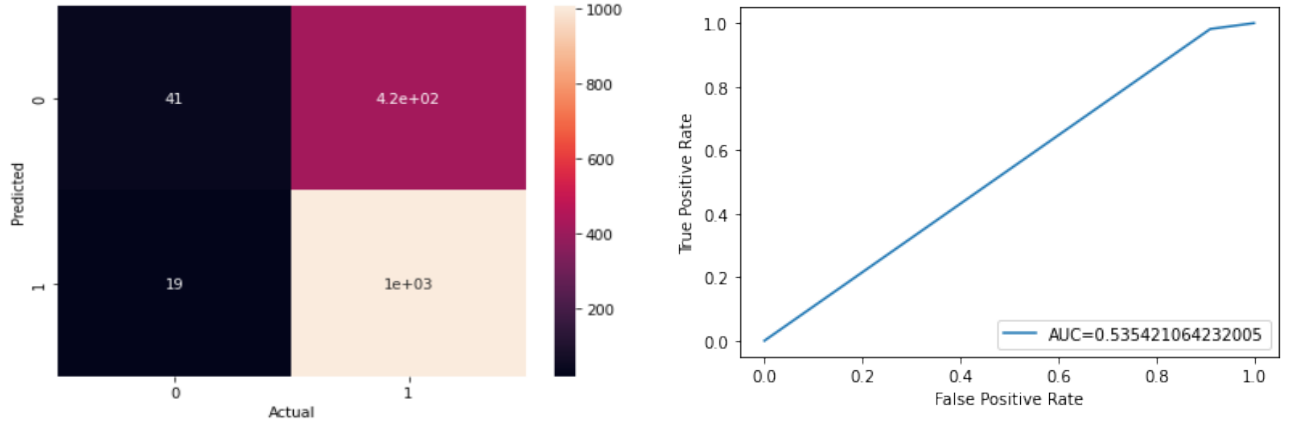
Benign-f1score=2*0.633*0.248/(0.633+0.248)=0.313968/0.881=0.356,

Malicious-f1score=2*0.736*0.936/(0.736+0.936)=1.377792/1.672=0.824. As we can see in Table 4-46, 4-47 and 4-48 that we get accuracy when K=1, we get **0.71**; when K=2, we get **0.36**; when K=3, we get **0.72**. It can be found in KNN ELF files have been analyzed by the sandbox static features shown at Figure 4-56(a), starting from K=3, shows that the f1-score is about 0.7 steadily.



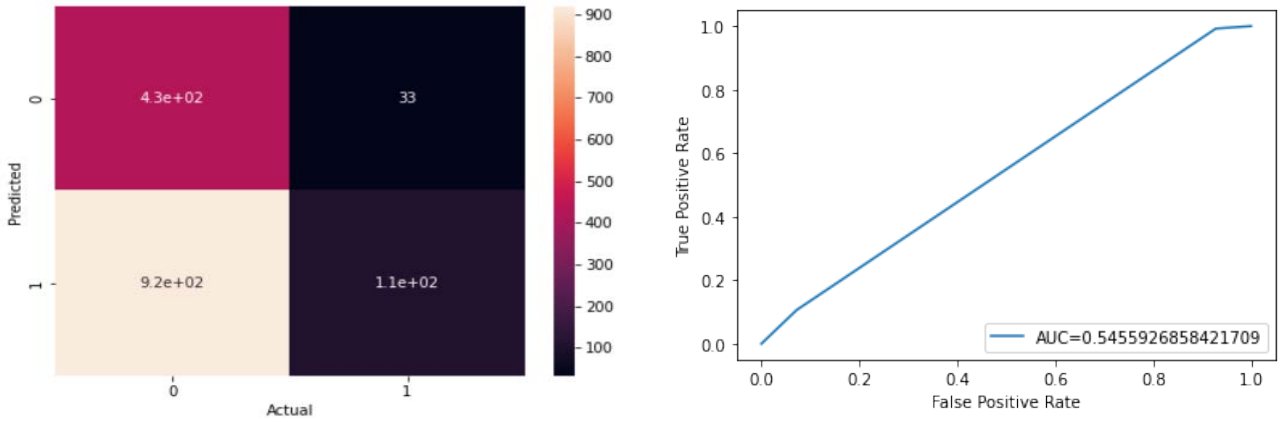
(a). (b).

Figure 4-56. KNN analyzed statically for system call features with different k values (a) F1-score (b) Error rate.



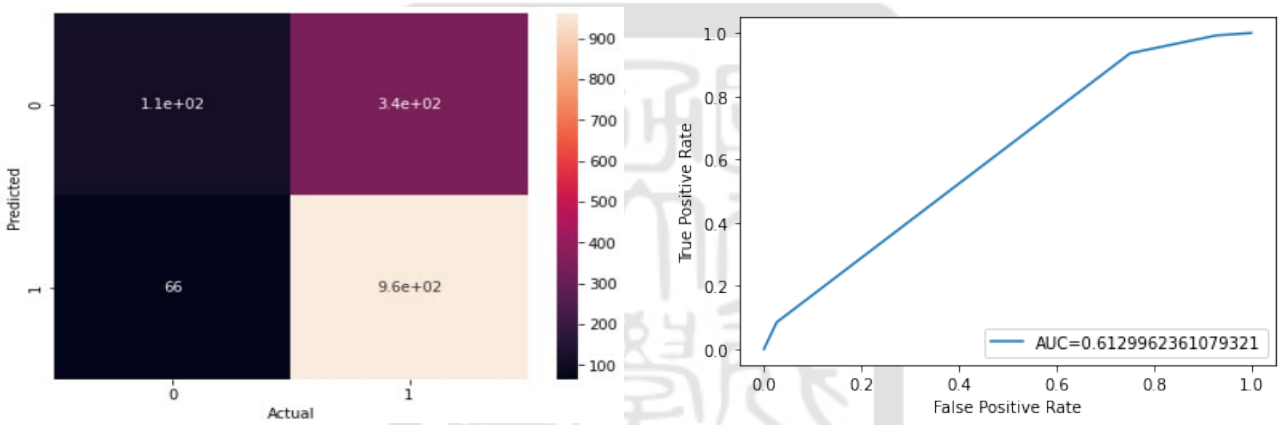
(a). (b).

Figure 4-57. KNN analyzed statically for system call features at k=1. (a) confusion matrix (b) ROC AUC graph.



(a). (b).

Figure 4-58. KNN analyzed statically for system call features at k=2. (a) confusion matrix (b) ROC AUC



(a). (b).

Figure 4-59. KNN analyzed statically for system call features at k=3. (a) confusion matrix (b) ROC AUC

Table 4-46. KNN model performance analyzed statically system call features at k=1.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.68	0.09	0.16	0.089	459
Malicious	0.71	0.98	0.82	0.982	1028
macro avg	0.70	0.54	0.49	**0.71**	1487
weighted avg	0.70	0.71	0.62	**0.71**	1487

Table 4-47. KNN model performance analyzed statically system call features at k=2.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.32	0.93	0.47	0.928	459
Malicious	0.77	0.11	0.19	0.106	1028
macro avg	0.54	0.52	0.33	**0.36**	1487
weighted avg	0.63	0.36	0.27	**0.36**	1487

Table 4-48. KNN model performance analyzed statically system call features at k=3.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.63	0.25	0.36	0.248	459
Malicious	0.74	0.94	0.82	0.936	1028
macro avg	0.68	0.59	0.59	**0.72**	1487
weighted avg	0.70	0.72	0.68	**0.72**	1487

Table 4-49. Machine learning model analyzed statically performance.

Decision tree	gini	**0.72**
Logistic regression		**0.69**
Random forest	gini	**0.72**
Gaussian naïve bayes		**0.68**
Support vector machine	SVC	**0.70**
K nearest neighbor	K=1	**0.71**
	K=2	**0.36**
	K=3	**0.72**

In Table 4-49, we organize the accuracy obtained from the machine learning experiments in Experiments using strace into a table. It can be found that the accuracy of all machine learning models are not very effective.

4.3.7 Convolutional Neural Network

In method 2, after we use **strace** to extract the feature values of each ELF file, we normalize all the feature values and convert them into 14*14 grayscale images. Figure 4-60 shows one of our ELF files converted into grayscale image, which is aa-enabled. We examine the results of convolutional neural network model when trained on system call features. Our results are shown in Figure 4-61 and

Table 4-50. In Table 4-50, Benign-precision=15/(15+0)=1.00, Benign-recall=15/(15+466)=0.031, Malicious-precision=1006/(466+1006)=0.683, Malicious-recall=1006/(0+1006)=1.00, macroavg-precision=(1.00+0.68)/2=0.84, macroavg-recall=(0.03+1.00)/2=0.52, weightedavg-precision=(1.00*481+0.68*1006)/1487=0.79, weightedavg-recall=(0.03*481+1.00*1006)/1487=0.69, Benign-f1score=2*1.00*0.031/(1.00+0.031)=0.062/1.031=0.060, Malicious-f1score=2*0.683*1.00/(0.683+1.00)=1.366/1.683=0.812. As we can see in Table 4-50 that we get accuracy of **0.68**.

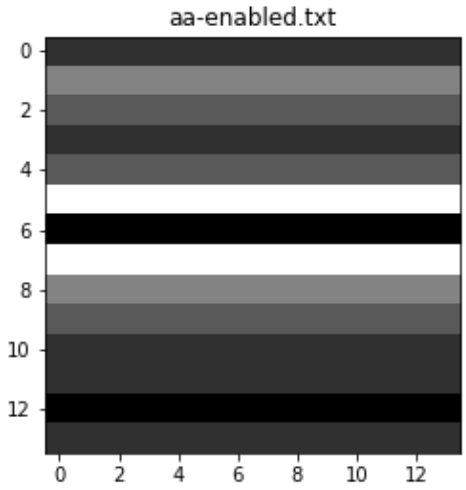


Figure 4-60. one of benign ELF files 14*14 grayscale image.

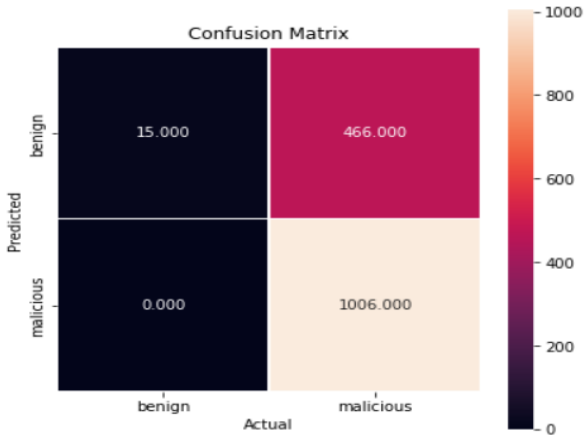
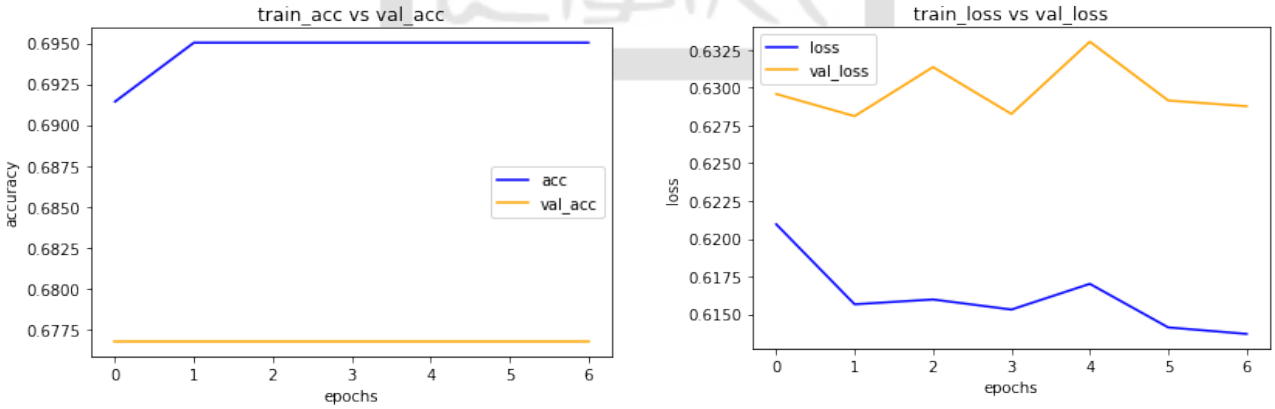


Figure 4-61. CNN analyzed statically confusion matrix.

Table 4-50. CNN model performance analyzed statically system call features.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	1.00	0.03	0.06	0.031	481
Malicious	0.68	1.00	0.81	1.000	1006
macro avg	0.84	0.52	0.44	**0.69**	1487
weighted avg	0.79	0.69	0.57	**0.69**	1487



(a). (b).

Figure 4-62. CNN analyzed statically training and validation (a) accuracy (b) validation loss.

4.4 Experiments using image technique

In method 3, we convert all the ELF files (bytes sequence) to 2D array and examine the results of convolutional neural network model when trained on ELF files transform to black and white images directly. Figure 4-63 shows one of our ELF files converted into black and white image, which is aa-enabled. Our results are shown in Figure 4-81 and Table 4-51. In Table 4-51, Benign-precision=318/(318+28)=0.919, Benign-recall=318/(318+148)=0.682, Malicious-precision=997/(148+997)=0.871, Malicious-recall=997/(28+997)=0.973, macroavg-precision=(0.92+0.87)/2=0.89, macroavg-recall=(0.68+0.97)/2=0.83, weightedavg-precision=(0.92*466+0.87*1025)/1487=0.88, weightedavg-recall=(0.68*466+0.97*1006)/1487=0.88, Benign-f1score=2*0.919*0.682/(0.919+0.682)=1.253516/1.601=0.783, Malicious-f1score=2*0.871*0.973/(0.871+0.973)=1.694966/1.844=0.919. As we can see in Table 4-51 that we get accuracy of **0.88**.

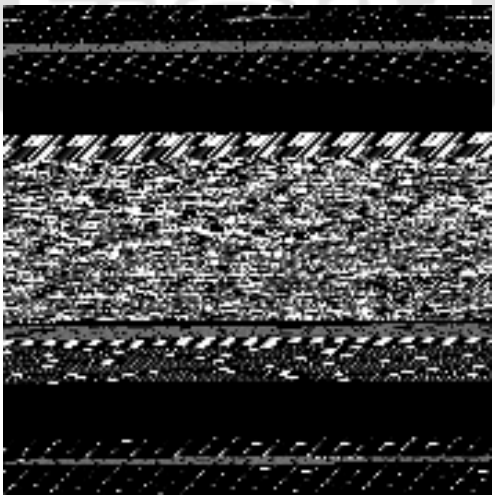


Figure 4-63. one of benign ELF files black and white image.

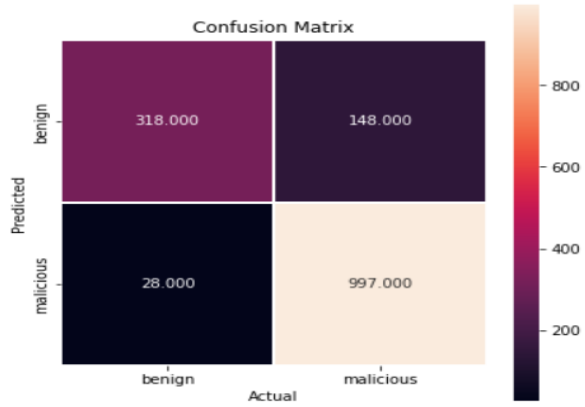
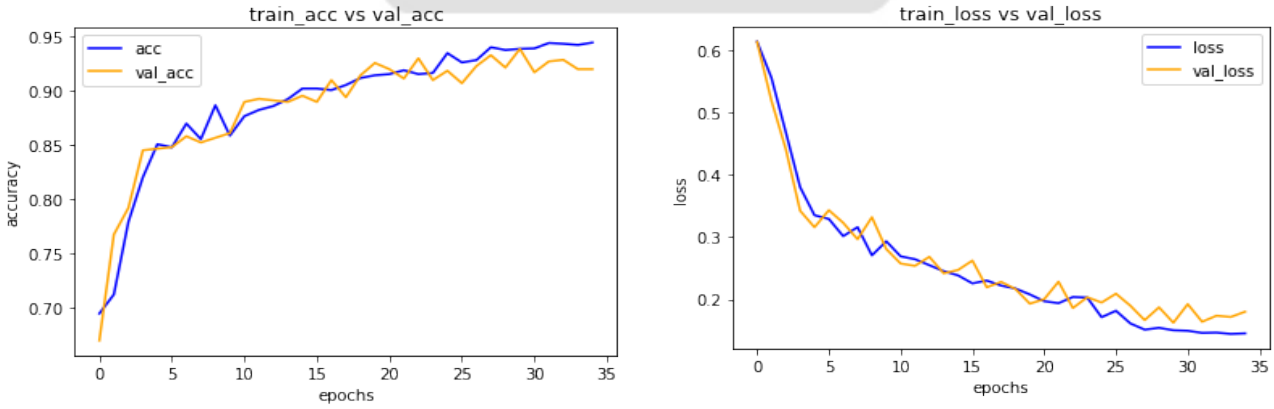


Figure 4-64. CNN ELF files transform to gray scale images directly confusion matrix.

Table 4-51. CNN model performance ELF files transform to gray scale images directly.

	precision	recall	f1-score	accuracy	# of testing ELF files
Benign	0.92	0.68	0.78	0.682	466
Malicious	0.87	0.97	0.92	0.973	1025
macro avg	0.89	0.83	0.85	**0.88**	1491
weighted avg	0.88	0.88	0.88	**0.88**	1491



(a). (b).

Figure 4-65. CNN ELF files transform to gray scale images directly training and validation (a) accuracy (b) validation loss.

Chapter 5 Conclusions and Future Works

The first method takes about 2-3 weeks to analyze all ELF files through **limon sandbox** on Ununtu20.04. The second method using **strace** to extract system calls of all ELF files takes about 3-4 hours. The third method directly convert all ELFs into black and white images within about 1 hour. When these three methods generate reports and extract features to consolidate them into csv files. We then use machine learning to train these data within 5 to 10 minutes, while using deep learning to train these data takes about 1-2 hours. Although sandbox has functions such as security, consistency, low installation complexity, less space occupied by the hard disk, and computer memory configuration, compared with the other two methods, sandbox is more time-consuming. In the first method, the accuracy of machine learning decision tree and random forest and deep learning CNN can reach more than 0.9. In the second method, the accuracy effects of all machine learning models and deep learning CNN are very poor. In the third method, the accuracy of only using deep learning CNN can reach 0.88. All three methods have the results by using CNN model. The first method is very time-consuming because it needs to collect all feature values before training. Therefore, if the experiment time is insufficient, the third method can be used to analyze the ELF file. In other words, if the experiment time is sufficient, we can use the first method to analyze the ELF files.

Although the decision tree and random forest models used in this thesis can effectively predict viruses and achieve extremely high accuracy, there are still some problems with these methods. Our sample files are already being determined if they are benign files or malicious files. But in practice, when we receive an IoT file, it is not easy to know whether these files are benign or malicious. Also, many malicious files must be run with some types of input data to produce malicious effects. In our experiments, our methods run these malwares without given any input data or parameters. Therefore, it is very likely that many malicious files will be judged as benign files in practice. In future work, we can add parameters to each file to fully execute these malwares.

References

[1] “Fake Websites Used in COVID-19 Themed Phishing Attacks, Impersonating Brands Like Pfizer and BioNTech,” https://unit42.paloaltonetworks.com/covid-19-themed-phishing-attacks/

[2] https://www.ithome.com.tw/news/112848

[3] “The security and privacy issues that come with the Internet of Things,” https://www.insiderintelligence.com/insights/iot-security-privacy/

[4] “Decision Tree,” https://www.ibm.com/topics/decision-trees.

[5] “Logistic Regression,” https://www.ibm.com/topics/logistic-regression.

[6] “Random Forest,” https://www.ibm.com/topics/random-forest.

[7] “Gaussian Naïve Bayes,” https://www.upgrad.com/blog/gaussian-naive-bayes/.

[8] “Support Vector Machine,” 2018, https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[9] “K-nearest neighbor,” https://www.ibm.com/topics/knn.

[10] “Convolutional Neural Network,” https://www.ibm.com/topics/convolutional-neural-networks.

[11] “Evaluation Metrics for Machine Learning – Accuracy, Precision, Recall, and F1 Defined,” 2020, https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec

[12] “How to Calculate & Use the AUC Score,” 2020, https://towardsdatascience.com/how-to-calculate-use-the-auc-score-1fc85c9a8430

[13] ”Linux Malware,” 2013, https://www.dropbox.com/sh/t8qak9zwfg45hva/AABOtR_L_k3KLIS4tPwIDwbBa/CLEAN_ELF_LINUX_50_files.zip?dl=0

[14] ”Linux Malware,” 2016, https://www.dropbox.com/sh/t8qak9zwfg45hva/AACxqfaDj05GPIOFqiDGZDhxa?dl=0&preview=MALWARE_ELF_LINUX_100_files.zip

[15] ”Virusshare,” 2014, https://ia801204.us.archive.org/view_archive.php?archive=/12/items/virusshare_malware_collection_000/VirusShare_ELF_20140617.zip

[16] ”Linux Malware,” 2014, https://github.com/MalwareSamples/Linux-Malware-Samples

[17] “Strace tool,” 2019, https://sourceforge.net/projects/strace/files/strace/4.18/strace-4.18.tar.xz/download?use_mirror=gigenet.

[18] InetSim, “Inetsim,” https://www.inetsim.org/packages.html, 2020, accessed: 2020-06-01.

[19] Linux, “Linux readelf,” https://linux.die.net/man/1/readelf, 2009, accessed: 2020-06-01.

[20] Misha Mehra, Jay N. Paranjape and Vinay J. Ribeiro, “Improving ML Detection of IoT Botnets using Comprehensive Data and Feature Sets”, *2021 13th International Conference on Communication Systems & Networks (COMSNETS)*.

[21] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim, and J. N. Kim, “An in-depth analysis of the mirai botnet,” in *2017 International Conference on Software Security and Assurance (ICSSA)*, Altoona, Pennsylvania, USA, 2017, pp. 6–12.

[22] Monappa, “Setting up limon sandbox for analysing linux binaries,” https://cysinfo.com/setting-up-limon-sandbox-for-analyzing-linux-malwares/, 2016, accessed: 2020-06-01.

[23] T. N. Phu, K. H. Dang, D. N. Quoc, N. T. Dai, and N. N. Binh, “A novel framework to classify malware in mips architecture-based iot devices,” *Security and Communication Networks*, 2019.

[24] K. A. Asmitha and P. Vinod, “A machine learning approach for linux malware detection,” in 2014 *International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*. Ghaziabad, India: IEEE, 2014, pp. 825–830.

[25] N. Koroniotis, N. Moustafa, E. Sitnikova, and J. Slay, “Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques,” in *International Conference on Mobile Networks and Management*. Melbourne, VIC, Australia: Springer, 2017, pp. 30–44.

[26] H. Nguyen, Q. Ngo, and V. Le, “Iot botnet detection approach based on psi graph and dgcnn classifier,” in *2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP).* Singapore: IEEE, 2018, pp. 118–122.

[27] T. Phu, L. Hoang, N. Toan, N. Tho, and N. Binh, “Cfdvex: A novel feature extraction method for detecting cross-architecture iot malware,” *SoICT 2019: Proceedings of the Tenth International Symposium on Information and Communication Technology*, pp. 248–254, 12 2019.

[28] M. Mehra and D. Pandey, “Event triggered malware: A new challenge to sandboxing,” in *2015 Annual IEEE India Conference (INDICON)*. New Delhi, India: IEEE, 2015, pp. 1-6.

[29] Safa Rkhouya, Khalid Chougdali, “Malware Detection Using a Machine-Learning Based Approach,” in *2021 International Journal of Information Technology and Applied Sciences*.

[30] J. Su, V. D. Vasconcellos, S. Prasad, S. Daniele, Y. Feng, and K. Sakurai, “Lightweight classification of iot malware based on image recognition,” in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. Tokyo, Japan: IEEE, 2018.

[31] Mohannad Alhanahnah, Qicheng Lin, Qiben Yan, Ning Zhang, Zhenxiang Chen, “Efficient Signature Generation for Classifying Cross-Architecture IoT Malware,” in *2018 IEEE Conference on Communication and Network Security (CNS).*

[32] Satish Pokhrel, Robert Abbas, Bhulok Aryal,” IoT Security: Botnet detection in IoT using Machine learning,” arXiv preprint arXiv:2104.02231, 2021.

[33] Huy-Trung Nguyen, Quoc-Dung Ngo, Doan-Hieu Nguyen, Van-Hoang Le, “PSI-rooted subgraph: A novel feature for IoT botnet detection using classifier algorithms,” *ICT Express*, pp. 128–138, 2020.