Machine Learning For Malware Detection Using P.E File N-Grams and Deep Learning on Executables Abdul Nasir Muniru Xian University of Technology

分类号	
学校代码	10700
学　　号	2181227006

西安理工大学

MASTER'S THESIS

(Academic Degree)

MACHINE LEARNING FOR CYBERSECURITY: IMPLEMENTATION OF MALWARE DETECTION USING P.E FILE, N-GRAMS AND DEEP LEARNING ON EXECUTABLES

ABDUL NASIR MUNIRU

学 科 门 类：	**Computer Engineering**
一 级 学 科：	**Computer Science and Technology**
二 级 学 科：	**Computer Application Technology**
指 导 教 师：	**Professor Sun Qindong**
申 请 日 期：	**2021 年 6 月**

论文题目：网络安全机器学习：利用 P.E 文件、N-Grams 和可执行文件的深度学习实现恶意软件检测。

学科名称：计算机应用技术

研 究 生：ABDUL NASIR MUNIRU **签 名：** ABDUL NASIR MUNIRU

指导教师：孙钦东 教授 **签 名：** [illegible]

摘 要

随着网络威胁和攻击越来越普遍，并迅速演变，支持恶意软件和反对恶意软件开发商之间的战斗似乎远远没有接近尾声。此外， 人们对于远程和虚拟通信计算机和电子系统的需求，随着 covid-19 和社会距离的加大而变得越来越大，从而推动了需求曲线向上。这为网络罪犯创造了机会，并导致恶意软件攻击数量显著增加。现代的恶意软件采用各种复杂和不受怀疑的新手段来获得访问，并使被攻击计算机系统不被发现。但是，使用经典的方法来处理恶意软件检测，如基于签名， AV 扫描，和其他方法来防止恶意软件攻击已被证明是过时的，不再有效。因此，处理恶意软件是计算机科学与工程，电子通信和技术发展的世界主要挑战之一。执法人员、商业实体和一般用户等利益攸关方遭受了多次严重损失。数据科学在网络安全中的重要性，在预测未来可能的威胁以减轻和预防恶意软件攻击方面，永远都不会被高估。数据科学、机器学习和人工智能在恶意软件预防和控制方面的一些应用已经在学术界提出并在现实世界中实现。我们的工作集中在一个探索性的实现机器学习的恶意软件检测算法，该算法失踪了 PE 文件 DLL 信息， N-Grams ， 和深度学习技术。我们使用公共恶意软件数据集进行实验，并详细介绍了 ML 技术和算法的概述和工作流程的恶意软件检测。并且，数据科学在网络安全中的应用的短暂下降也被探讨.

关键词：恶意软件检测；机器学习；网络安全；深耕；数据科学；

Title: Machine Learning for Cybersecurity: Implementation of Malware Detection Using P.E File, N-Grams and Deep Learning on Executables.

Major: Computer Application Technology

Name: Abdul Nasir Muniru **Signature:** Abdul Nasir Muniru

Supervisor: Prof. Sun Qindong **Signature:** Qindong Sun

Abstract

As cyberthreats and attacks are increasingly pervasive, constantly, and rapidly evolving, the battle between pro-malware and anti-malware developers seems to be nowhere close to the end. Also, Covid-19 pandemic and social distancing drives the need for computer and electronic systems for remote and virtual communication which has exponentially influence and boosted the demand curve for computing resources and computer devices upwards, this created an opportunity for cybercriminals leading to a significant increase in malware attacks. Modern-day malware takes various sophisticated and unsuspecting novel means to gain access and attack computer systems without detection. Therefore, the classical ways of dealing with malware detection such as; signature-based, AV scanning, and other ways of preventing malware attacks have proven to be obsolete and no longer efficient. Due to this challenge, dealing with malware is one of the major challenges to the world of computer science and engineering, electronic communication, and technology development. Stakeholders such as law enforcement agents, business entities, and general users have suffered several severe losses. The significance of data science in cybersecurity can never be overrated when it comes to predicting possible future threats for the mitigation and prevention of malware attacks. Several applications of data sciences, machine learning, and Artificial Intelligence in the prevention and control of malware have been proposed in the academic world and implemented in the real world. Our work in this paper is focused on an exploratory implementation of Machine Learning for malware detection leveraging PE file DLL information, N-Grams, and Deep Learning Techniques. We used public malware data sets for our experiments and also detailed the overview and workflow of ML techniques and algorithms for malware detection. The short fall in the usage of data science in cybersecurity has also been explored.

Key words: Malware Detection; Machine Learning; Cybersecurity; Deep Leaning; Data Science;

目 录

1 INTRODUCTION

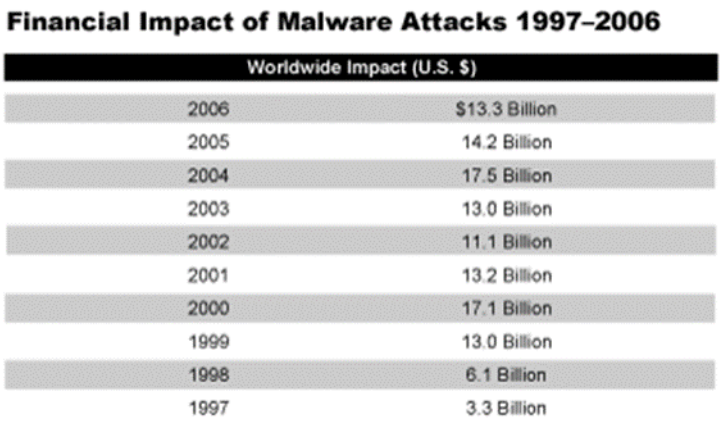
In this section, we introduce our research background, the status of research work, the main work done, innovation and structure, and significance. It summarizes the background of our study and gives an overview of the threats and relevance of malware detection in cybersecurity.

1.1 Research Background and Significance

Computers and information systems are inevitable in this era of human evolution, hence the title "Computer era"[1] Covid-19 has further escalated the need to digitalizing our world. Computers are widely used in all aspects of life including personal, cooperation, and government businesses. Cyber threats are a serious issue of concern to all stakeholders in the Computing and electronic Technology business because it contributes to a huge number of losses to individuals and organizations. Cyber threats cut across all spheres of the digital world, from virtual systems to physical systems. Sun et al reiterated the need to improve the security and quality of real-time multimedia transmission in cyber-physical-social systems[2]

Malware is described by Moir as "a short for malicious software and is typically used as a catch-all term to refer to any software designed to cause damage to a single computer, server, or computer network, whether it's a virus, spyware, et al."[3] also Christodorescu et al described a malware instance as a program that has malicious intent[4]. Malware forms a major part of cybersecurity threats. Various types of Malware include but are not limited to spyware, rootkit, adware, worm, virus, ransomware, and cryptomining malware. With the exponential growth in the use of cloud computing, so there is an increasing number of malware creations due to enormous big data deposits created in cloud servers. Malware attacks can break down critical infrastructure, compromise sensitive data, hijack and abuse digital, and computational resources as well as evade communication networks.

Apart from the year 2005 which recorded a decline, malware attacks have recorded high financial impacts since 1970 when the earlier documented virus appeared[5]. Figure 1 shows the financial impact of malware attacks from 1997-2006.



Financial Impact of Malware Attacks 1997–2006

Worldwide Impact (U.S. $)	
2006	$13.3 Billion
2005	14.2 Billion
2004	17.5 Billion
2003	13.0 Billion
2002	11.1 Billion
2001	13.2 Billion
2000	17.1 Billion
1999	13.0 Billion
1998	6.1 Billion
1997	3.3 Billion

Fig.1-1 Financial Impact Malware Attacks 1997-2006[6]

In as much as the anti-malware community is doubling its efforts, malware developers are never relenting their efforts too. In 2010 about less than 50 million unique malicious executables were recorded, the figure was doubled to about 100 million in 2012, by 2019 it moved to more than 900 million, and currently in 2021, 1165 million and still increasing[7].

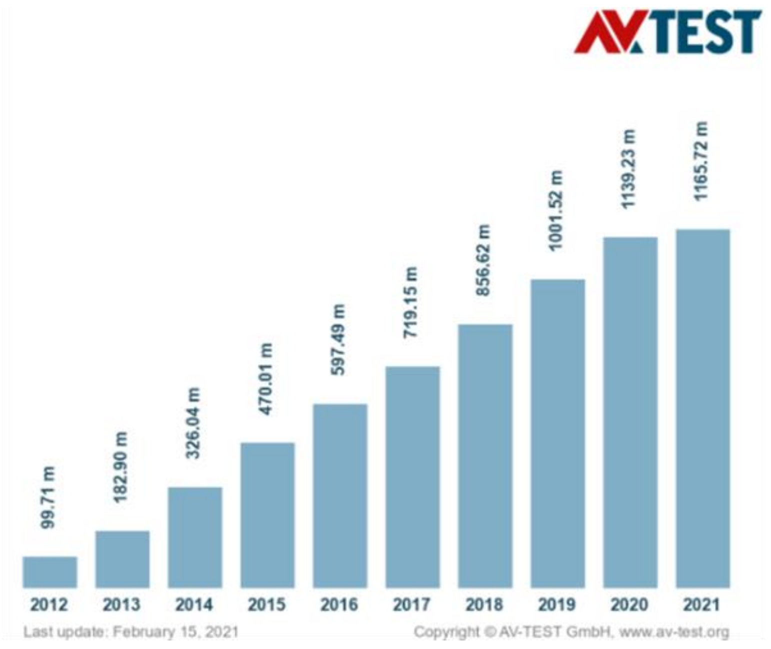


Fig.1-2 AV-Test Institute Malware statistics[7]

Traditional signature-based anti-virus systems fail to detect polymorphic and new, previously unseen malicious executables[8]. Several proposals and programs have been made by experts of the anti-malware community for the detection, prevention, and mitigation of malware. Techniques such as the Heuristic Approach, Sandbox Approach, Signature-Based Detection, and Integrity Checking in addition to the use of recent technologies such as machine learning and deep learning models although have been very useful in the fight, detection, and prevention of malware attacks, the battle between pro-malware and anti-malware developers seem to be far from over.

1.2 Research Status of Malware Detection

1.2.1 Usage of Machine Leaning for Malware Detection

Gibert et al indicated “The earliest documented virus appeared during the 1970s. It was known as the Creeper Worm and was an experimental self-replicating program that copied itself to remote systems and displayed the message: “I’m the creeper, catch me if you can”. Later, in the early 80s, appeared Elk Cloner, a boot-sector virus that targeted Apple II computers. From these simple beginnings, a massive industry was born and, since then, the fight against malware has never stopped. By the looks of it, this fight turned out to be a never-ending and cyclical arms race: as security analysts and researchers improve their defenses, malware developers continue to innovate, find new infection vectors and enhance their obfuscation techniques. Malware threats continue to expand vertically (i.e., numbers and volumes) and horizontally (i.e., types and functionality) due to the opportunities provided by technological advances. Internet, social networks, smartphones, IoT devices and so on, make it possible for the creation of smart and sophisticated malware.”[9].

Classical methods of detecting malware such as signature-based are limited in performance due to scalability[10], also static code analysis could have done a better job because it uses disassembling mechanism hence it can achieve complete coverage, however, it falls short since code obfuscation prevents the unpacking and decrypting of the executable files before the performance of the analysis[11][12]. To cover the shortfall of the static code analysis, dynamic code analysis is proposed to make it unnecessary for the unpacking or decrypting of the executable file which does not only resource consuming but also time-intensive since it takes place in a virtual environment[13].

The figure below presents the differences between the classical traditional and modern-day advanced malware features.

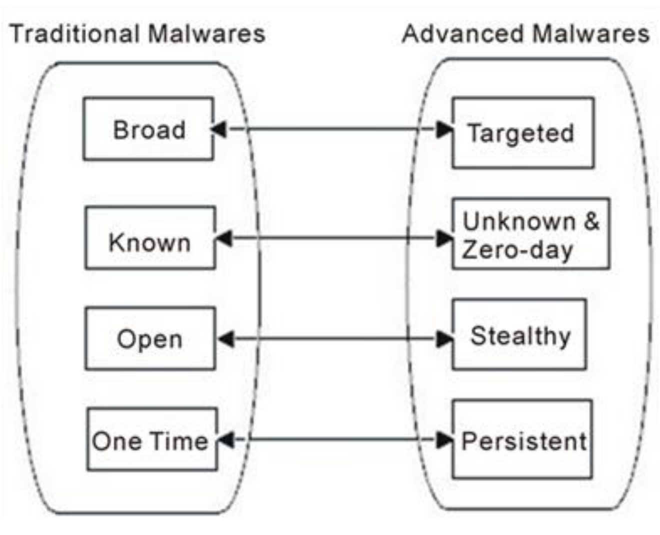


Fig.1-3 Traditional Vs. Advanced malware[11]

As Ekta et al stated Malware is one of the most terrible and major security threats facing the Internet today. According to a survey, conducted by FireEye in June 2013, 47% of the organizations experienced malware security incidents/network breaches in the past year. The malware is continuously growing in volume (growing threat landscape), variety (innovative malicious methods), and velocity (fluidity of threats). These are evolving, becoming more sophisticated, and using new ways to target computers and mobile devices[11].

1.3 Main Work and Innovation

Our work is mainly an exploratory implementation of machine learning techniques for malware detection using three mechanisms; The P.E files DLL, N-grams and Deep Learning on executable samples. In this paper we demonstrate the practical application and implementation of Machine Learning techniques for the detection and classification of malware, using both supervised and unsupervised machine learning techniques for the classification of malware leveraging cybersecurity data science. We emphasize the description of machine learning workflow and methods of their usage in the context of cybersecurity data science.

The main contribution of our work is summarized as follows;

1) Demonstrated the application of three (3) different machine learning techniques to classify and predict malware.

2) Discuss the Machine Learning technique for Malware Classification

3) Examine the previous works done on the research on malware detections in the direction of a data-driven approach in the domain of cybersecurity.

4) Present a comprehensive guide of the framework and workflow of machine learning in the context of cybersecurity data science for malware detection.

1.4 Research Content and Organizational Structure

Our work is mainly an exploratory implementation of machine learning for malware detection.

The entire work is divided into six chapters and the outline is as follows;

Chapter One: Introduction. This chapter mainly describes the background and significance of machine learning for malware detection. The status of malware detection at Home and abroad is also described in detail where we place much focus on the inefficiencies of the classical methods of malware detection and prevention while emphasizing the new paradigm of cybersecurity data science where machine learning techniques are deployed in malware detection and classification. We then give the innovation points and finally describe comprehensively our approach and arrangement of this thesis.

Chapter Two: Related Works and Literature Review. This chapter mainly focuses on relevant research literature and other previous works that have been conduction relating to malware detection and machine learning. We go deep into a review of malware evolution taking a view of the history and phases of transitions malware development and prevention of malware have been through. We also discuss the various types of malwares and their origination. We then take a deep look into Machine Learning and Cybersecurity which is the new paradigm in the fight and prevention of malware. Furthermore, we detailed the relationship between Data Science and Cybersecurity stating the roles each plays in the fight against and prevention of malware.

Chapter Three: Elements of Malware Detection in Machine Learning. This chapter is mainly dedicated to the discussion of all relevant factors that are required for a successful implementation of the machine learning technique for malware detection. We give an overview of the static and dynamic malware analysis. We also discuss in detail the classical techniques of malware detection. The PE file which is a critical component for building features of a successful machine learning model is also discussed laying out its features. We also present a Machine learning workflow where we give an overview of each of the stages in the basic workflow of every machine learning setup process. We move further to discuss machine learning techniques for cybersecurity. Finally, we presented the differentiation between Machine Learning, Deep Learning, and Neural Network.

Chapter Four: Experimentation. In this chapter we performed three main experiments; PE file DLL, N-grams, and Deep Learning machine learning techniques to perform classification of malware through the labeling of benign and malicious files.

Chapter Five: Experiment Discussion and Result Analysis. This chapter discusses the experiment setup and results from the analysis.

Chapter Six: Conclusion

This is the concluding chapter. We present a summary of all the work we have done in this paper. We also presented the prospects of our work and future expectation.

1.5 Summary

This chapter gives an introduction to our work background and the significance of using machine learning for malware detection. It also presented the scheme and structure of our work.

2 RELATED WORKS AND LITERATURE REVIEW

In this section, we present and discuss the lots of works that have been done in the area of malware detection, machine learning, and data science with cybersecurity. We cover a view of related surveyed research, literature, and works done on malware and machine learning for cybersecurity.

2.1 Malware Evolutions

For the past decades, Cybersecurity professionals and industry players have been battling the menace of cyber threats and attacks from malware. Numerous anti-malware programs have been created to aid in the fight against malware attacks. Due to the complex and sophisticated nature of modern-day malware, conventional anti-malware programs are unable to detect and prevent metamorphic malware from execution. Also, the proliferation of malware has presented a serious threat to the security of computer systems.

First Generation Malware: structure of the malware does not change[14]. In 1986, the first generation of malicious code was comprised of DoS viruses, which infected the operating system and programs of a PC[15].

Second Generation Malware: second generation, the internal structure of malware changes in every variant while the actions are maintained the same. Based on how variances are created in malware, second-generation malware is further classified as Encrypted, Oligomorphic, Polymorphic, and Metamorphic Malware[14].

Third Generation Malware: The introduction of high-impact, high-profile mass-mailer worms marked the beginning of the third generation of malicious code: Melissa (1999), “I Love You” (2000), Anna Kournikova (2001), SoBig (2003), and Mydoom (2004). The highly prolific network worms, such as Code Red (2001), SQL Slammer (2003), Blaster (2003), and Sasser (2003) are also indicative of this generation[15].

Fourth Generation Malware: The last three generations of malicious code authors wrote and distributed malicious code primarily to receive praise from peers and to gain notoriety. However, as we’ve entered the fourth generation, it has become clear that code authors are not looking for bragging rights, but rather cash—and lots of it[15].

2.1.1 Malware Types

Although different kinds and types of malware are still being developed. Below is the list of major types of malware[16];

(1) Virus: a small program designed to cause trouble by gaining access to your device. It can copy your data or slow your device down. A virus spreads by duplicating and attaching itself to

other files[17].

(2) Cryptojacking is the unauthorized use of someone else's computer to mine cryptocurrency. Hackers do this by either getting the victim to click on a malicious link in an email that loads cryptomining code on the computer or by infecting a website or online ad with JavaScript code that auto-executes once loaded in the victim's browser[18]. The figure below shows some common cryptomining malware detected globally.

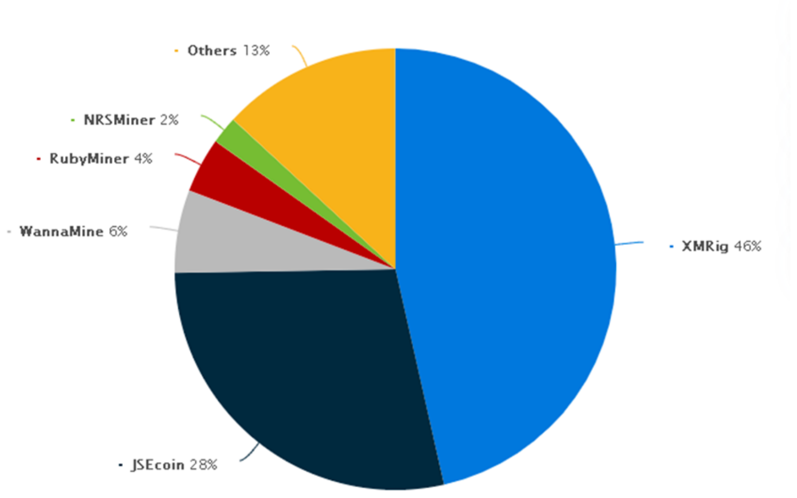


Fig.2-1 Most commonly detected cryptomining malware families affecting corporate networks worldwide from January to June 2020[19]

(3) Adware: Generically adware is a software application in which advertising banners are displayed while any program is running. Adware can automatically get downloaded to your system while browsing any website and can be viewed through pop-up windows or through a bar that appears on a computer screen automatically. Adwares are used by companies for marketing purposes[16].

(4) Spyware: Spyware is software that installs itself onto devices and then steals personal information about the user, like passwords, email addresses, and other important information. It often does it just by keeping a record of everything the user types, which is called key logging. Some spyware can even use your webcam without your knowledge[17].

(5) Scareware: Many desktop users have encountered scareware, which attempts to frighten the victim into buying unnecessary software or providing their financial data. Scareware pops up on a user's desktop with flashing images or loud alarms, announcing that the computer has been infected. It usually urges the victim to quickly enter their credit card data and download a fake antivirus program[20].

(6) Fileless malware: Unlike traditional malware, fileless malware does not download code onto a computer, so there is no malware signature for a virus scanner to detect. Instead, fileless malware operates in the computer's memory and may evade detection by hiding in a trusted utility,

productivity tool, or security application. An example is Operation RogueRobin, which was uncovered in July 2018. RogueRobin is spread through Microsoft Excel Web Query files that are attached to an email. It causes the computer to run PowerShell command scripts, providing attackers access to the system. As PowerShell is a trusted part of the Microsoft platform, this attack typically does not trigger a security alert. Some fileless malware is also clickless, so a victim does not need to click on the file to activate it[20].

(7) Trojan: A Trojan horse is not a virus. It is a destructive program that looks like a genuine application. Unlike viruses, Trojan horses do not replicate themselves but they can be just as destructive. Trojans also open a backdoor entry to your computer which gives malicious users/programs access to your system, allowing confidential and personal information to be theft. Example: - JS.Debeski.Trojan.

(8) Trojan horses are broken down in classification based on how they infect the systems and the damage caused by them. The seven main types of Trojan horses are:

a) Remote Access Trojans(RAT): A malicious program that remotely accesses infected resources. They are a class of malware that gives an attacker direct, interactive access to a victim's personal computer, allowing the attacker to steal private data stored on the machine, spy on the victim in real-time using the camera and microphone, and interact directly with the victim via a dialog box.

b) RATs have been used for surveillance, information theft, and extortion of victims[21][22]. RAT capabilities usually include program installation and removal, file manipulation, reading data from the keyboard, webcam hijacking, and clipboard monitoring.

c) Data Sending Trojans: This is a kind of malicious software that aims at stealing sensitive data from victims' hosts, and most of which are made up of a client, always a controller and a server, always as a controlee[23]. They supply the attacker with sensitive data, such as log files, passwords, e-mail addresses, credit card information, or IM contact lists. These Trojans can look for pre-defined data, for example, only passwords or credit card information, or they can install a keylogger and send all recorded keystrokes back to the attacker[24].

d) Destructive Trojans: designed to destroy data stored on the victim's computer[25].

e) Proxy Trojans: Trojan horse that uses the victim's computer as a proxy server, providing the attacker an opportunity to execute illicit acts from the infected computer, like banking fraud, and even malicious attacks over the internet[25].

f) FTP Trojans: This type of Trojan horse uses port 21 to enable the attackers to connect to the victim's computer using File Transfer Protocol[25].

g) Security software disabler Trojans: This Trojan horse is designed to disable security software like firewall and antivirus, enabling the attacker to use many invasion techniques to invade the victim's computer, and even to infect more than the computer.

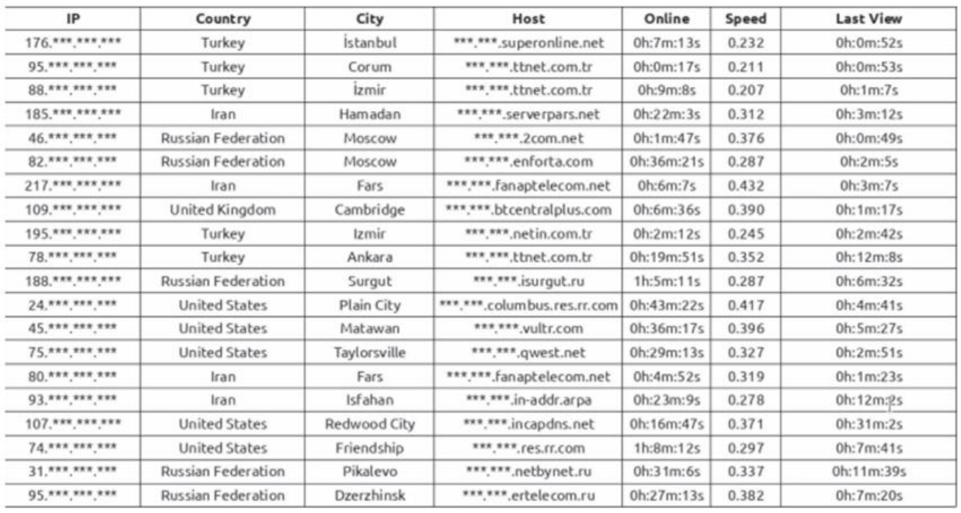
h) Denial-of-service attack Trojans: Trojan horse designed to give the attacker opportunity to realize Denial-of-Service attacks from the victim's computer[25].

(9) Warms: are malicious programs that make copies of themselves again and again on the local drive, network shares, etc. The only purpose of the worm is to reproduce itself again and again. It doesn't harm any data/file on the computer. Unlike a virus, it does not need to attach itself to an existing program. Worms spread by exploiting vulnerabilities in operating systems. Examples of worms are W32.SillyFDC.BBY, Packed.Generic.236 and W32.Troresba. Due to its replication nature, it takes a lot of space in the hard drive and consumes more CPU uses which in turn makes the pc too slow also consumes more network bandwidth[16].

(10) Sniffers: refers to the monitoring of internet traffic in real-time. Sniffers are programs or hardware devices that can spy on you and all of your internet activity. Sometimes legitimate, sometimes criminal, sniffers can leave you feeling exposed. Read on to understand what sniffers do, how they work, and how to protect yourself against sniffing attacks from hackers [26].

(11) Keyloggers: is a type of spyware that can be used to track and log the keys you strike on your keyboard, capturing any information typed. Keyloggers are insidious because you don't know they're there, watching and recording everything you type. Keyloggers can be software or hardware. Software is more common. The majority of keyloggers are used to capture payment card data you might enter online with a device. Once captured, your data can be retrieved by the person on the other end of the keylogger program. Many keyloggers have root-kit functionality. That means they're hiding in your system. These so-called Trojan-spy programs can track user activity — including keystrokes and screenshots - save the data to your hard disk, and forward the information to cybercriminals[27].

(12) Botnets(short for "robot network"): is a network of computers infected by malware that are under the control of a single attacking party, known as the "bot-herder." Each machine under the control of the bot-herder is known as a bot. From one central point, the attacking party can command every computer on its botnet to simultaneously carry out a coordinated criminal action. The scale of a botnet (many comprised of millions of bots) enables the attacker to perform large-scale actions that were previously impossible with malware. Since botnets remain under the control of a remote attacker, infected machines can receive updates and change their behavior on the fly. As a result, bot-herders are often able to rent access to segments of their botnet on the black market for significant financial gain[28].



IP	Country	City	Host	Online	Speed	Last View
176.***.***.***	Turkey	İstanbul	***.***.superonline.net	0h:7m:13s	0.232	0h:0m:52s
95.***.***.***	Turkey	Corum	***.***.ttnet.com.tr	0h:0m:17s	0.211	0h:0m:53s
88.***.***.***	Turkey	İzmir	***.***.ttnet.com.tr	0h:9m:8s	0.207	0h:1m:7s
185.***.***.***	Iran	Hamadan	***.***.serverpars.net	0h:22m:3s	0.312	0h:3m:12s
46.***.***.***	Russian Federation	Moscow	***.***.2com.net	0h:1m:47s	0.376	0h:0m:49s
82.***.***.***	Russian Federation	Moscow	***.***.enforta.com	0h:36m:21s	0.287	0h:2m:5s
217.***.***.***	Iran	Fars	***.***.fanaptelecom.net	0h:6m:7s	0.432	0h:3m:7s
109.***.***.***	United Kingdom	Cambridge	***.***.btcentralplus.com	0h:6m:36s	0.390	0h:1m:17s
195.***.***.***	Turkey	Izmir	***.***.netin.com.tr	0h:2m:12s	0.245	0h:2m:42s
78.***.***.***	Turkey	Ankara	***.***.ttnet.com.tr	0h:19m:51s	0.352	0h:12m:8s
188.***.***.***	Russian Federation	Surgut	***.***.isurgut.ru	1h:5m:11s	0.287	0h:6m:32s
24.***.***.***	United States	Plain City	***.***.columbus.res.rr.com	0h:43m:22s	0.417	0h:4m:41s
45.***.***.***	United States	Matawan	***.***.vultr.com	0h:36m:17s	0.396	0h:5m:27s
75.***.***.***	United States	Taylorsville	***.***.qwest.net	0h:29m:13s	0.327	0h:2m:51s
80.***.***.***	Iran	Fars	***.***.fanaptelecom.net	0h:4m:52s	0.319	0h:1m:23s
93.***.***.***	Iran	Isfahan	***.***.in-addr.arpa	0h:23m:9s	0.278	0h:12m:2s
107.***.***.***	United States	Redwood City	***.***.incapdns.net	0h:16m:47s	0.371	0h:31m:2s
74.***.***.***	United States	Friendship	***.***.res.rr.com	1h:8m:12s	0.297	0h:7m:41s
31.***.***.***	Russian Federation	Pikalevo	***.***.netbynet.ru	0h:31m:6s	0.337	0h:11m:39s
95.***.***.***	Russian Federation	Dzerzhinsk	***.***.ertelecom.ru	0h:27m:13s	0.382	0h:7m:20s

Fig.2-2 A screenshot of available botnets in an underground market where digital cash could be spent[29].

(13) Spamware: is a software utility designed specifically by spammers for spamming. Spamware enables a user to search, sort, and compile a list of email addresses and provides an automated email broadcasting solution. This can be used to send spam or unsolicited emails to unsuspecting recipients[30]. Not all software that sends a bulk email is spamware, as email listserver software can be used for legitimate purposes, rather than spam.

(14) Ransomware: holds your PC hostage and demands money. It locks up your computer, threatening to wipe all your data, and demanding payment for the release of your data, files or to regain the ability to use your computer again[31].

2.2 Machine Leaning and Cyber security Paradigm

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy[32]. Generally, machine learning assumes that the future will behave like the past, but this is not always true, this implies machine learning can only recognize things it has seen before since it can only memorize patterns in the training data.

In recent days, cybersecurity is undergoing a massive shift in technology and its operations in the context of computing, and data science (DS) is driving the change, where machine learning (ML), a core part of "Artificial Intelligence" (AI) can play a vital role to discover the insights from data. Machine Learning can significantly change the cybersecurity landscape[32]. Below is a figure showing the popularity of Machine Learning, Data Science, and Cybersecurity[32].

Figure 2-3 Popularity trends of data science, machine learning, and cybersecurity over time, where the x-axis represents the timestamp information and the y-axis represent the corresponding popularity values[32].

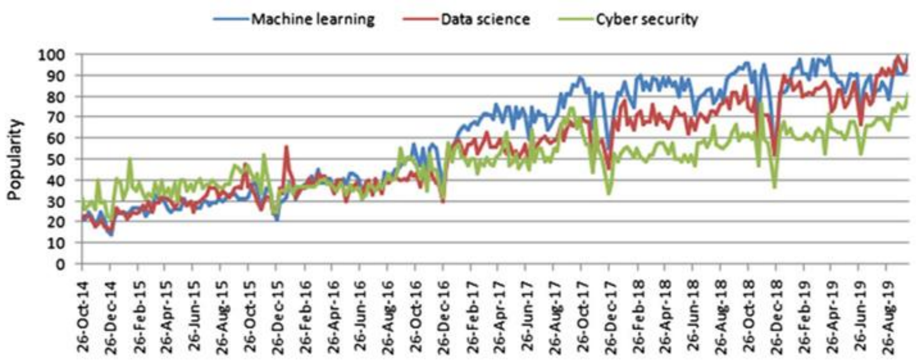


Fig.2-3 Popularity trends of data science, machine learning, and cybersecurity over time, where the x-axis represents the timestamp information and the y-axis represents the corresponding popularity values[32]

One of the pioneers in the field of applying machine learning to malware detection is Schultz et al, in the year 2000, they presented a data mining framework that detects new, previously unseen malicious executables accurately and automatically. The data mining framework automatically found patterns in their data set and used these patterns to detect a set of new malicious binaries. Comparing their detection methods with a traditional signature-based method, their method more than doubles the current detection rates for new malicious executables[32]. Gavriluţ et al proposed a versatile framework in which one can employ different machine learning algorithms to successfully distinguish between malware files and clean files while aiming to minimize the number of false positives. In their paper, they presented the ideas behind their framework by working firstly with cascade one-sided perceptrons and secondly with cascade kernelized one-sided perceptrons. After having been successfully tested on medium-size datasets of malware and clean files, the ideas behind their framework were submitted to a scaling-up process that enables them to work with very large datasets of malware and clean files[33].

Sharma et al presented Detection of Advanced Malware by Machine Learning Techniques in which they studied the frequency of opcode occurrence to detect unknown malware by using a machine learning technique[34]. In their work, they also highlighted the top 10 malware of the windows operating system as indicated in the diagram below.

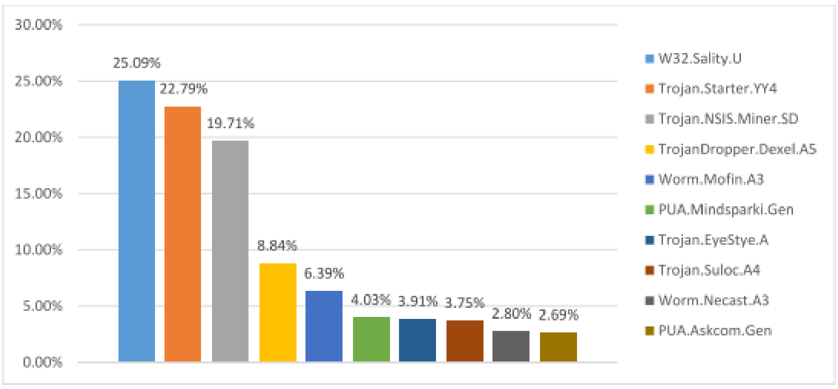


Fig.2-4 Top Ten Windows malware[34]

Yanfang et al developed an Intelligent Malware Detection System (IMDS) using objectiveoriented Association (OOA) mining-based classification through relying on the analysis of Windows API execution sequences called by PE files.[8]. Their system is an integrated system consisting of three major modules: PE parser, OOA rule generator, and rule-based classifier. They adopted the OOA Fast FPGrowth algorithm is adapted to efficiently generate OOA rules for Classification. Further more they made a comprehensive experimental study on a large collection of PE files obtained from the anti-virus laboratory of KingSoft Corporation to compare various malware detection approaches. Their IMDS outperformed anti-virus software such as Norton AntiVirus and McAfee VirusScan and previous data mining detection systems which employed Naïve Bayes, Support Vector Machine (SVM), and Decision Tree techniques[8]. The structure of their work is presented in the figure below.

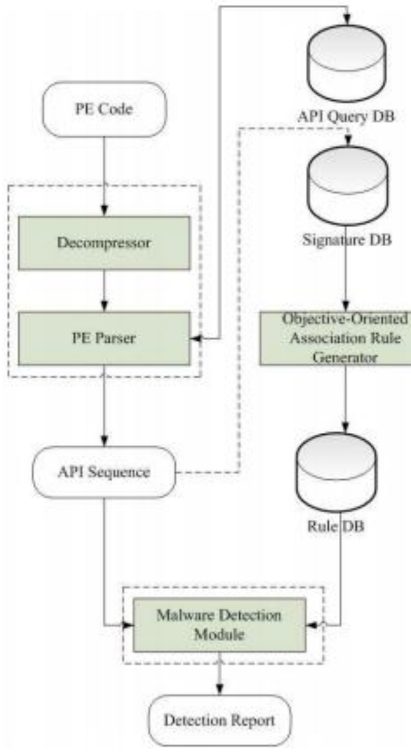


Fig.2-5 IMDS system Architecture[8]

A complimentary work was done by Yanfang et al to automatic malware categorization using cluster ensemble. In their paper, they established the need for the automation of malware classification since many of the existing techniques isolated successes with limited effectiveness and their efficiency and few have been applied in the real anti-malware industry. Their Automatic Malware Categorization System(AMCS) relied on instruction frequency and function-based instruction sequences. Their AMCS automatically grouped malware samples into families that share some common characteristics using a cluster ensemble by aggregating the clustering solutions generated by different base clustering algorithms.

The main contribution of their work is to overcome the instability of clustering results and improve clustering performance. They also develop new base clustering algorithms to account for the different characteristics of feature representations and propose a novel cluster ensemble framework for combining individual clustering solutions. The figure below shows the architecture of the AMCS.

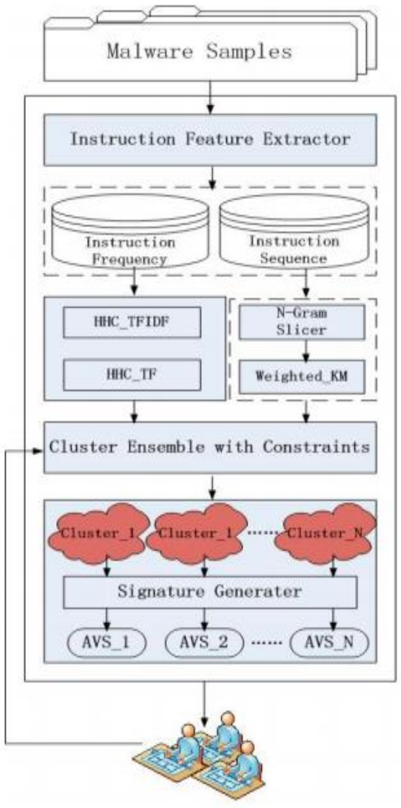


Fig.2-6 The system Architecture of AMCS[35]

The ACMS system has the following major traits:

(1) Well-Chosen Feature Representations: Instruction frequency and function-based instruction sequences are used as malware feature representations. These instruction-level features well represent variants of malware families and can be efficiently extracted. In addition, these features can be naturally used to generate signatures for malware detection.

(2) Carefully-Designed Base Clusterings: The choice of base clustering algorithms is largely dependent on the underlying feature distributions. To deal with the irregular and skewed distributions of instruction frequency features, we propose a hybrid hierarchical clustering algorithm that combines the merits of hierarchical clustering and k-medoids algorithms. To identify the hidden structures in the subspace of function-based instruction sequences, we use a weighted subspace K-medoids algorithm to generate base clusterings.

(3) A Principled Cluster Ensemble Scheme: Our AMCS system uses a cluster ensemble scheme to combine the clustering solutions of different algorithms. Our cluster ensemble scheme is a principled approach based on the consensus partition and can utilize the domain knowledge in the form of sample-level constraints.

(4) Human-in-the-Loop: In many cases, the domain knowledge and expertise of virus analysts can greatly help improve the categorization results. Our AMCS system offers a mechanism to incorporate the domain knowledge in the form of sample-level constraints (such as, some file samples are variants of a single malware, or some file samples belong to different malware types).

(5) Natural Application for Signature Generation: The categorization results generated by our AMCS system can be naturally used to generate signatures for each malware family. These signatures are very useful for malware detection[35].

Xinbo et al propose a novel method of malware detection using data visualization and adversarial training on ML-based detectors to efficiently detect the different types of malware and their variants. Their experimental results on the MS BIG malware database and Ember database demonstrate that their proposed method can prevent the zero-day attack and achieve up to 97.73% accuracy, along with 96.25% on average for all the malware tested. Their work proposes the first ML-based malware visualization detection method with a suitable model regularization by exploiting both AE and AT techniques. Furthermore, their proposed method can generate the variance and camouflage of malware to mitigate the novel and probable malware in real detection. This method (Visual-AT) is also a more accurate and efficient detection method to prevent zero-day attacks. Lastly, their work carries out performance analysis of the proposed method and evaluation in terms of accuracy and robustness of real dataset[10]. The figure below shows the process of malware visualization transformation.

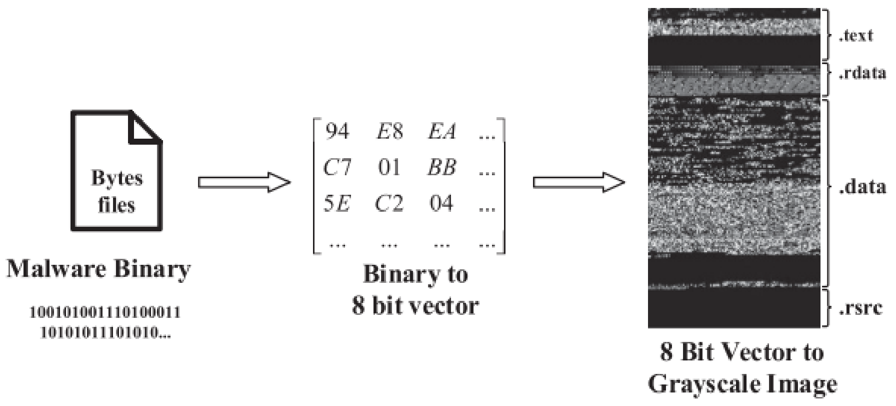


Fig.2-7 Process of malware visualization transformation[10]

Hansen et al also suggested the use of Genetic programming for the prevention of cyberterrorism through dynamic and evolving intrusion detection. To investigate this proposition, they experimented using a very large dataset from the 1999 Knowledge Discovery in Database (KDD) Cup data, supplied by the Defense Advanced Research Projects Agency (DARPA) and MIT's Lincoln Laboratories. Using machine-coded linear genomes and a homologous crossover operator in genetic programming, promising results were achieved in detecting malicious intrusions. The resulting programs execute in real-time, and high levels of accuracy were realized in identifying both positive and negative instances[36].

Yin et al explored the modeling of intrusion detection based on deep learning and then propose a deep learning approach for intrusion detection using recurrent neural network (RNN-IDS). Their work undertook to study the performance of their model in binary classification and multiclass classification, and then the number of neurons and different learning rate impacts on the performance of the proposed model. They further compared their work with those of J48, artificial neural networks, random forest, support vector machine, and other machine learning methods proposed by previous researchers on the benchmark data set. The experimental results show that RNN-IDS is very suitable for modeling a classification model with high accuracy and that its performance is superior to that of traditional machine learning classification methods in both binary and multiclass classification. The RNN-IDS model improves the accuracy of intrusion detection and provides a new research method for intrusion detection[37]. The figure below is a block diagram of the proposed RNN-IDS.

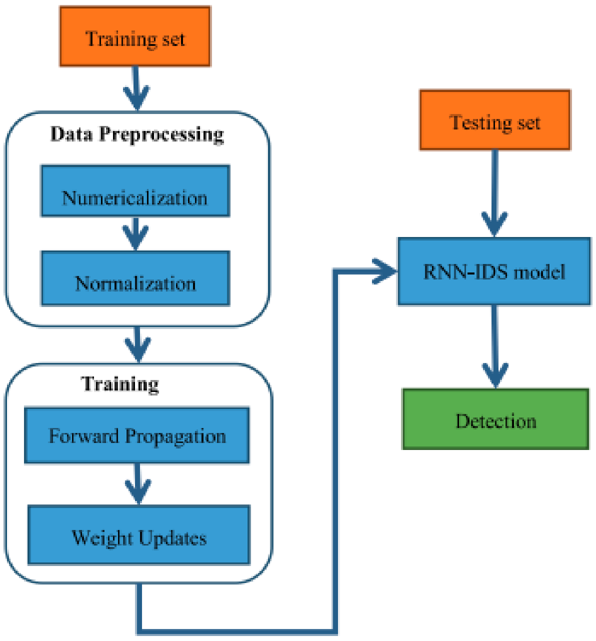


Fig.2-8 Block Diagram of Proposed RNN-IDS[37]

In the fields of security of IoT devices which are very critical elements of powering the massive growth and success of data collection and cloud computing, Almiani et al presented an artificially full-automated intrusion detection system for Fog security against cyber-attacks. Their proposed model uses a multi-layered of recurrent neural networks design to be implemented for Fog computing security that is very close to the end-users and IoT devices. They moved further to demonstrate their proposed model using a balanced version of the challenging dataset: NSL-KDD[38].

Furthermore, in the interest of data science cybersecurity for malware detection, the popularity of recurrent networks (RNN) in recent days has influenced their use in malware detection and classification. Tobiyama et al proposed a malware process detection method based on process behavior in the possible infected terminal. In their paper, they investigated the stepwise application of Deep Neural Networks to classify malware process behavior in possible infected terminals. They first trained the Recurrent Neural Network(RNN) to extract features of process behavior and then trained the Convolutional Neural Network (CNN) to classify feature images that are generated by the extracted features from the trained RNN. The evaluation results in several image sizes by comparing the AUC of obtained ROC curves and they obtained AUC= 0:96 in best case[39].

Although machine learning has been a great tool in the fight against malware attacks, it has some bottlenecks and threats it does face. ML for malware detection is vulnerable to adversarial attacks[40]. Anderson et al summarized the various attacks that have been proposed for machine learning models in information security, each of which requires the adversary to have some degree of knowledge about the model under attack. Importantly, even when applied to attacking machine learning malware classifier based on static features for Windows portable executable (PE) files, these attacks, previous attack methodologies may break the format or functionality of the malware[41]. To counter the challenging task of detecting advanced malware variants i.e. polymorphic and metamorphic variations, Choi et al introduced AMVG: Adaptive Malware Variant Generation Framework Using Machine Learning to study bypassing malware detection methods efficiently[42].

In all indications, the marriage of data science and cybersecurity is playing a significant role in the cyberwar fight. Sarker et al presented an overview of cybersecurity data science from the machine learning perspective where they focus and briefly discussed cybersecurity data science from where the data is gathered analyzed and patterns are driven from them and provision of more effective security solutions[43].

2.2.1 Attacking Machine Leaning Models

Anderson et al categorized attacking machine learning into three. They include:

(1) Direct gradient-based attacks: in this attack, the model must be fully differentiable and the structure and weights must be known by the attacker. Given this, the attacker can essentially query the model directly to determine how best to bypass it.

(2) Attacks against models that report a score; The attacker has no knowledge about the model structure, but has unlimited access to probe the model and may be able to learn how to decrease the score.

(3) Binary black-box attacks. The attacker has no knowledge about the model but has unlimited access to probe the model[41].

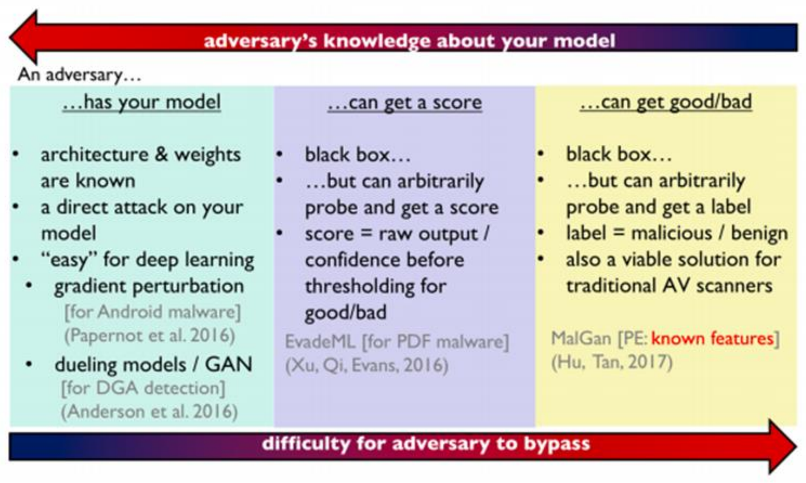

adversary's knowledge about your model
An adversary…
…has your model
architecture & weights are known
a direct attack on your model
"easy" for deep learning
gradient perturbation
[for Android malware] (Papernot et al. 2016)
dueling models / GAN
[for DGA detection] (Anderson et al. 2016)
…can get a score
black box…
…but can arbitrarily probe and get a score
score = raw output / confidence before thresholding for good/bad
EvadeML [for PDF malware] (Xu, Qi, Evans, 2016)
…can get good/bad
black box…
…but can arbitrarily probe and get a label
label = malicious / benign
also a viable solution for traditional AV scanners
MalGan [PE: known features] (Hu, Tan, 2017)
difficulty for adversary to bypass

Fig.2-9 Attack categories, categorized into three coarse bins[41]

2.3 Data Science Vs Cybersecurity

The significant role played by data science in cybersecurity in the fight against malware and other cybercrimes is worth the description of data science as one of the best and efficient techniques in the fight against malware. Sarker et al stated that cybersecurity is undergoing massive shifts in technology and its operations in recent days, and data science is driving the change. Extracting security incident patterns or insights from cybersecurity data and building a corresponding data-driven model, is the key to make a security system automated and intelligent[43].

2.3.1 Data Science

Data science is an interdisciplinary field aiming to turn data into real value. It is the study of the generalizable extraction of knowledge from data. Data may be structured or unstructured, big or small, static or streaming. The value may be provided in the form of predictions, automated decisions, models learned from data, or any type of data visualization delivering insights. Data science includes data extraction, data preparation, data exploration, data transformation, storage and retrieval, computing infrastructures, various types of mining and learning, presentation of explanations and predictions, and the exploitation of results taking into account ethical, social, legal, and business aspects[44][45].

2.3.2 Cybersecurity

Cyber Security involves reducing the risk of a malicious attack on software, computers, and networks. This includes tools used to detect break-ins, stop viruses, block malicious access, enforce authentication, enable encrypted communications, and on and onAlso Cybersecurity is a set of technologies and processes design to protect computers, networks, programs, and data from attack, damage, or unauthorized access[46].

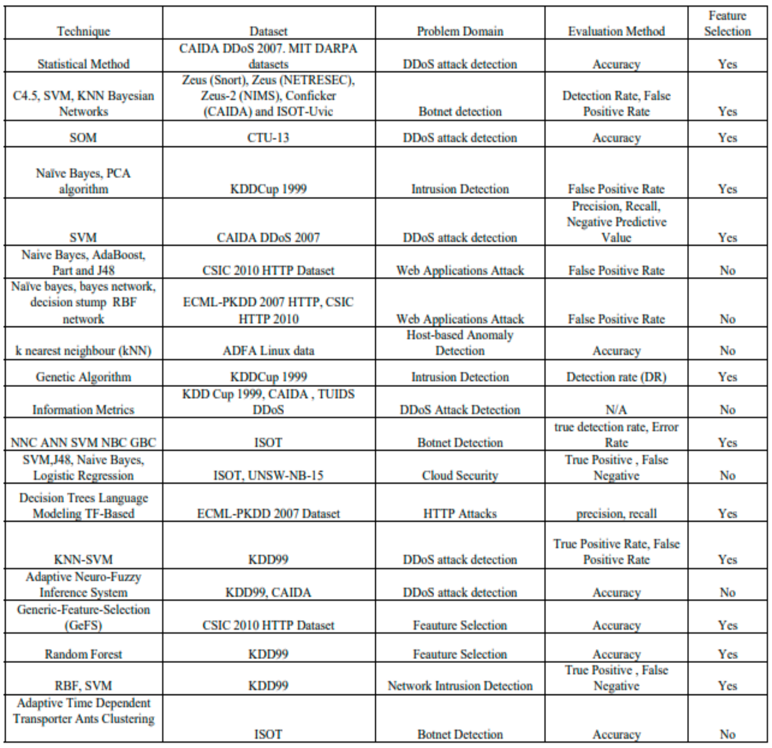
2.3.3 Cybersecurity Data Science

This is a branch of Data Science where cybersecurity data is used to train a model for searching and detecting cyber-attacks. Data Science is transforming the world, and today most industries are data-driven. So, it is critically important for the future of intelligent cybersecurity systems and services because "security is all about data". When we seek to detect cyber threats, we are analyzing the security data in the form of files, logs, network packets, or other relevant sources. Traditionally, security professionals didn't use data science techniques to make detections based on these data sources. Instead, they used file hashes, custom-written rules like signatures, or manually defined heuristics [46][47].

2.3.4 Cybersecurity Data

Cyber-security Data is a collection of raw data which contains various types of cyberattacks and their relevant features. The reason for collecting the raw security data is to analyze the various patterns of security incidents or malicious behavior and to build a data-driven security model to achieve the desired security for our data. Several datasets exist in the area of cybersecurity including intrusion analysis, malware analysis, anomaly, fraud, or spam analysis that is used for various purposes[48].

Tab.2-1 Cyber Security Data Sets Publicly Available[48]



Technique	Dataset	Problem Domain	Evaluation Method	Feature Selection
Statistical Method	CAIDA DDoS 2007. MIT DARPA datasets	DDoS attack detection	Accuracy	Yes
C4.5, SVM, KNN Bayesian Networks	Zeus (Snort), Zeus (NETRESEC), Zeus-2 (NIMS), Conficker (CAIDA) and ISOT-Uvic	Botnet detection	Detection Rate, False Positive Rate	Yes
SOM	CTU-13	DDoS attack detection	Accuracy	Yes
Naïve Bayes, PCA algorithm	KDDCup 1999	Intrusion Detection	False Positive Rate	Yes
SVM	CAIDA DDoS 2007	DDoS attack detection	Precision, Recall, Negative Predictive Value	Yes
Naive Bayes, AdaBoost, Part and J48	CSIC 2010 HTTP Dataset	Web Applications Attack	False Positive Rate	No
Naïve bayes, bayes network, decision stump RBF network	ECML-PKDD 2007 HTTP, CSIC HTTP 2010	Web Applications Attack	False Positive Rate	No
k nearest neighbour (kNN)	ADFA Linux data	Host-based Anomaly Detection	Accuracy	No
Genetic Algorithm	KDDCup 1999	Intrusion Detection	Detection rate (DR)	Yes
Information Metrics	KDD Cup 1999, CAIDA , TUIDS DDoS	DDoS Attack Detection	N/A	No
NNC ANN SVM NBC GBC	ISOT	Botnet Detection	true detection rate, Error Rate	Yes
SVM,J48, Naive Bayes, Logistic Regression	ISOT, UNSW-NB-15	Cloud Security	True Positive , False Negative	No
Decision Trees Language Modeling TF-Based	ECML-PKDD 2007 Dataset	HTTP Attacks	precision, recall	Yes
KNN-SVM	KDD99	DDoS attack detection	True Positive Rate, False Positive Rate	Yes
Adaptive Neuro-Fuzzy Inference System	KDD99, CAIDA	DDoS attack detection	Accuracy	No
Generic-Feature-Selection (GeFS)	CSIC 2010 HTTP Dataset	Feauture Selection	Accuracy	Yes
Random Forest	KDD99	Feauture Selection	Accuracy	Yes
RBF, SVM	KDD99	Network Intrusion Detection	True Positive , False Negative	Yes
Adaptive Time Dependent Transporter Ants Clustering	ISOT	Botnet Detection	Accuracy	No

2.4 Summary

This chapter gives an indebt description of previous research works that have been conducted similar to the topic of our main work on malware detection and machine learning. We present the evolution of malware giving summary details of malware types as well as machine learning paradigm and cybersecurity.

3 ELEMENTS OF MALWARE DETECTION IN MACHINE LEARNING

In this section, we present an overview and deeply discuss critical and fundamental elements needed to be taken cognizance of when using machine learning for malware detection. We also presented a general workflow to set up the foundational structure for the implementation of our framework for using machine learning for malware detection.

3.1 Static Vs Dynamic Malware Analysis

The two major components of malware analysis are Static and Dynamic malware analysis.

Static Malware Analysis; This involves examining the basic structure of the malware executable without executing it. It is independent of its environment and does not contaminate the environment during or after the analysis. The detection patterns used in the static analysis include string signature, byte-sequence n-grams, syntactic library call, control flow graph, and operational code(opcode) frequency distribution[11].

Dynamic Malware Analysis: This involves examining the behavior of the malware executable after executing it. the behavior of the malicious code (interaction with the system) is analyzed while it is being executed in a controlled environment (virtual machine, simulator, emulator, sandbox, etc.).

Generally, Dynamic analysis gives more insight into the malicious file than static analysis, although it requires running the file which may be problematic.

3.2 Malware Detection Techniques

A malware detector is a system that attempts to identify malware[4]. Aside from the mention approaches below, several other signs of progress have been made in the application of data mining and machine learning in the detection and mitigation of malware.

3.2.1 Heuristic Approach

This technique employs data mining and machine learning techniques to learn the behavior of an executable file[49]. The usage of the heuristic approach was first sited in the Works of Schult et al where they deployed Naïve Bayes and Multi Naïve Bayes for the classification of benign and malicious malware files[50]. In the heuristic approach, a set of features extracted are used in the process of analysis and classification. The table below shows the advantages and disadvantages of different kinds of features used in the heuristic approach of the malware detection technique.

Tab.3-1 Features used in Heuristic malware Detection[49]

Feature	Advantages	Disadvantage
API	1. Detects polymorphic and unknown malware. Fewer false positives than other scanners. Outperforms other classification approaches in both detection ratio and accuracy 2. Outperforms popular antivirus software tools, such as McAfee Virus Scan and Norton Antivirus as well as previous data-mining based detection systems. Reduces the number of Generated rules. Fewer detection time rather than other scanners. 3. Detects malware before their execution. Works well under code obfuscation methods. 4. Generates semantic signature. Categorizes API calls to 128 groups to reduce the graph size. Detects metamorphic malware. 5. Large and imbalanced gray list. Outperforms other classification methods in terms of performance and efficiency. Low time complexity.	1. Large set of generated rules for building classifier. 2. Only provides binary prediction 3. Large size of graph for comparison
OpCode	1. Shows the ability of single OpCodes to use as feature in machine learning malware detection techniques. 2. Detects obfuscated malware variants. 3. Extracts proper features for machine learning classifiers 4. Does not need high number of executables for each of the classes. 5. Suitable for Imbalance datasets. 6. Detects metamorphic malware. 7. Detects unknown malware. 8. Reduces the false positive rate.	1. High number of executables for each of the classes. 2. Imbalance datasets 3. High number of executables for each of the classes. 4. It is difficult to evaluate. May be biased.

表 3-1（续）

Feature	Advantages	Disadvantage
CFG	1. Detects metamorphic malwares. 2. High detection ratio. Low false positive rate.	1. Did not compare the efficiency of its algorithm with other techniques. 2. Did not evaluate false negatives.
CFG API Calls	1. Detects unknown malware. 2. Reaches better accuracy than CFG method. 3. Low false positive rate. 4. Less complexity in comparison with dynamic behavior method.	
Content-based behavior-based features	1. Improves the accuracy of malware detection effectively. 2. Outperforms all popular ensemble learning methods.	1. Takes a long time of training.
File relation	Reduces the false positive rate. 1. The accuracy of the file relation-based classifier is similar to the file content-based classifier, while combining these two features outperforms each of them.	
File content	1. Uses large and real datasets.	
Dependency graph	1. Outperforms antiviruses in detecting VBS malwares. 2. Outperforms existing anti-virus software's in Virus Total.	1. Test the algorithm on small dataset. 2. Not appropriate for PE files. 3. Do not detect all metamorphic techniques. 4. Computational cost of GA.
(PE) section plain-text strings byte sequence	1. Detects previously undetectable malicious executable. 2. Detects borderline binaries	

表 3-1（续）

Feature	Advantages	Disadvantage
N-Gram	1. Improves the accuracy of malware detection effectively. 2. Low false positive ratio.	3. Time Complexity

3.2.2 Sandbox Analysis Approach

This moves suspected files to a sandbox or secured environment to activate and analyze the file without exposing the rest of the network to potential risk. Kaspersky describes A sandbox as a system for malware detection that runs a suspicious object in a virtual machine (VM) with a fully-featured OS and detects the object's malicious activity by analyzing its behavior. If the object performs malicious actions in a VM, the sandbox detects it as malware. VMs are isolated from the real business infrastructure[51].

3.2.3 Signature-Based Detection

Signature-based methods use the patterns extracted from various malware to identify them and are more efficient and faster than any other methods. These signatures are often extracted with special sensitivity for being unique, so those detection methods that use this signature have a small error rate[49]. The main disadvantages of this method are, its inability to detect new variants of malware as well as huge resource requirements in the completion of its task. As indicated by J. Scott, Signature and behavioral-based anti-malware are no match for next-generation adversaries who utilize mutating hashes, sophisticated obfuscation mechanisms, self-propagating malware, and intelligent malware components. It is no longer enough to detect and respond[52].

3.2.4 Integrity Checking

The above-mentioned techniques are only good enough when dealing with a certain type of malware with no sophisticated structure i.e. First Generation malware. As indicated by Sanjay et al To counter/defend the malware, there are many anti-malware defense systems based on signature matching, heuristic analysis, normalization, and machine/deep learning. Some malware can be detected easily by signature matching anti-malware software. But the use of advanced obfuscation techniques signature matching techniques can't detect advanced/second generation or an unknown variant of malware [53]. But the use of advanced obfuscation techniques signature matching techniques can't detect advanced/second generation or an unknown variant of malware.

3.3 The Portable Executable (PE) File

We have earlier listed the PE section plain text string byte sequence as a feature used in the

Heuristic malware detection technique. As described by Singh, the PE stands for the portable executable file format. As the name suggests, the format can be portable across all 32-bit operating systems and can be executed on any version of windows[54]. It is also described as a file format for executables, object code, DLLs, and others used in 32-bit and 64-bit Windows operating systems[55]. Portable executable or PE file features play a key role in the detection of packed executables. Packing performs a lot of changes to the internal structure of PE files in such a way that it makes it very difficult for any Reverse Engineering Technique, Anti-Virus (AV) scanner, or similar kinds of programs to figure out whether the executable is malware or benign. Therefore, it is very important to figure out whether a given executable is packed or non-packed before detecting it as malicious or benign. Once a binary is detected as packed, it can be unpacked and can be given to AV or similar kinds of programs[56]. Choi et al proposed a PE File header Analysis-based Packed PE File Detection Technique (PHAD). Their proposed PHAD is based on a PE Header analysis, where the unusual characteristics often found in PE headers are utilized to detect packed files. To accomplish their task, they defined A Characteristics Vector(CV) that consists of eight elements, and the Euclidean distance (ED) of the files are calculated and represent the base threshold for the detection of packed file[57].

A.A Zaidan et al presented a New Technique of Hidden Data in PE-File with in Unused Area One where they proposed a new system of information hiding is presented. The proposed system aims to hide information (data file) in unused area 1 of any execution file (exe.file), to make sure changes made to the exe.file will not be detected by anti-virus and the functionality of the exe.file is still functioning. The system includes two main functions; first is the hiding of the information in the unused area 1 of PE-file (exe.file), through the execution of four processes (specify the cover file, specify the information file, encryption of the information, and hiding the information) and the second function is the extraction of the hiding information through three processes (specify the steno file, extract the information, and decryption of the information). The testing result shows; the result file does not make any conflict with anti-virus software and the exe.file still function as usual after the hiding process. The proposed system is implemented by using Java[58]. There are many different types of executable files, but for the purposes of this paper, we choose to concentrate on the use of the PE file. The figure below shows the structure of the PE file.

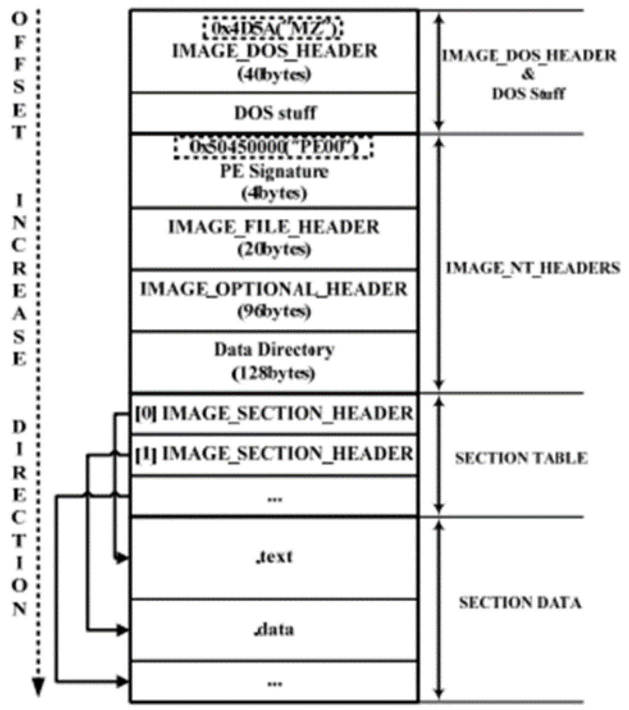


Fig.3-1 Windows PE file Format[59]

As shown in the figure above, the PE file is composed of four main parts, DOS header, NT header, Section Table, and Section Data. There are numerous attributes for PE file executions. The value of each attribute has its normal range, and some attributes are related to other attributes. Hence, one value of one attribute can be affected by the value of another attribute. In normal PE files, the relationship is very tight. For example, if the "Characteristics" attribute of a section has the "IMAGE_SCN_CNT_CODE" flag, it usually also has the "IMAGE_SCN_MEM_EXECUTE" flag, as the executable part is typically the code[59].

The table below shows PE file sections and their features:

Tab.3-2 PE File Items Descriptions

Section Item	Description
IMAGE_DOS_HEADER	Occupies the first 64 bytes of the PE file, consists of a Microsoft MS-DOS stub, the PE signature, the COFF file header, and an optional header. All MS-DOS-compatible executable files set this value to 0x54AD, which represents the ASCII characters MZ. MS-DOS headers are sometimes referred to as MZ headers for this reason. [60][61]
IMAGE_NT_HEADERS	Provides the pointer needed by the ImageRvaToVaQ for the PE header. [60]
SECTION TABLE	Each row of the section table is, in effect, a section header. This table immediately follows the optional header, if any.[60]

表 3-2（续）

Section Item	Description
SECTION DATA	The data for each section is located at the file offset that was given by the PointerToRawData field in the section header. The size of this data in the file is indicated by the SizeOfRawData field. If SizeOfRawData is less than VirtualSize, the remainder is padded with zeros.[60]

3.4 Machine Learning Workflow

Machine learning typically has an iterative workflow process, this involves obtaining the data, cleaning and preparing the data, building models, validating the models, and finally deploying the model to production. Before initiating any progress, a certain critical question needs to be answered. They include;

(1) What is the main objective? What prediction are you trying to make?

(2) What are the target features?

(3) What is the input data? Is it available?

(4) What kind of problem are you facing (Binary Classification/ Clustering)?

(5) What is the expected improvement?

(6) What is the current status of target features and how are they going to be measured? It is worth knowing that not every problem can be solved. Hypothesis such as; (I) Given the necessary inputs the needed output can be predicted. (II) There are sufficient information and data to learn the relationship between inputs and outputs.

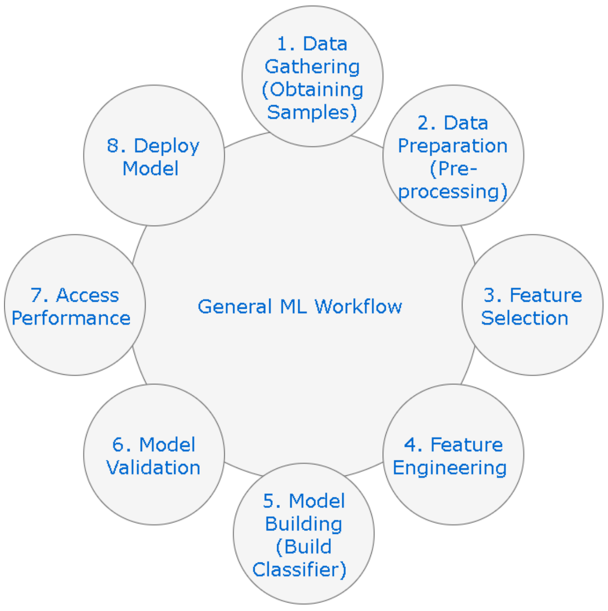


Fig.3-2 General Machine Learning workflow

Step 1: Data Gathering

Acquisition of data set is the genesis of every journey or adventure in building a machine learning model. The input variable or independent features are designated as X while the Output variables or dependent labels are designated as Y. The dataset is an A X B matrix where A = columns(features) and B= rows(samples). Data can be sourced from internet activities, database activities, and application activities, or any other data collection point. Data acquired can be very messy and full of errors, therefore, needs to be cleaned before usage. Raw security data can be collected from various cyber sources, these data can then be analyzed to draw various patterns of malicious security behaviors. Aside from malware analysis, several other analyses can be performed on datasets which include; fraud, spam, intrusion, and anomaly analysis.

Step 2: Data Preparation

After gathering the raw malware data, data preparation or pre-processing undertakes to deal with missing data, noisy data, extremely large values, and unorganized text data. Data preparation or preprocessing is considered a very important stage in the workflow of Machine Learning as it cleans and scrutinizes the data to standardize it. To an extent, it is advisable to spend 80% of time and effort in pre-processing while the remaining 20% perform the analysis. Otherwise, everything else will be wrong if bad data is being fed in, "Garbage in, Garbage out"- George Fuechsel

(1) Exploratory Data Analysis(EDA);

This is used as a preliminary step to familiarize with the data. EDA is an approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to maximize insight into a data set, uncover underlying structure, extract important variables, detect outliers and anomalies, test underlying assumptions, develop parsimonious models and determine optimal factor settings [62].

EDA is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions[63].

In a nutshell, it is performed to gain a preliminary understanding and allow us to get acquainted with the dataset.

(2) Dealing with Categorical Data

It is important to align categorical data very well to make its interpretation easy for the machine learning algorithm. Categorical data can be ordinal i.e can be sorted(cloth's size: Large/Medium/Small) or nominal i.e can't be sorted(cloth's color: red/yellow/blue) features.

a) Mapping ordinal feature: this is converting categorical string values into integers. Example Large:2, Medium:1, Small:0.

b) Encoding nominal features (one-hot encoding): this involves creating a new set of dummy features for each unique value in the nominal feature column. Example if red is selected, it is assigned as red=1,yellow=0,blue=0 for our set of colors.

Step 3: Feature Selection

Feature selection is the process to choose a subset of input variables by eliminating the noisy or redundant features and keeping the quality patterns[64]. It primarily focused on removing non-informative or redundant predictors from the model. This is one of the functions in machine learning which has a huge impact on the performance of a model. The irrelevant feature can negatively affect the model's performance by limiting training data, computational resources and confuse learners.

Supervised feature selection techniques can be used for labeled data while unsupervised techniques can be used for unlabelled data. These techniques can further be classified under the following ;

(1) Filter methods; uses the intrinsic properties of the features measured via univariate statistics instead of cross-validation performance. The selection of features is independent of the classifier[65].

(2) Wrapper methods; features are selected using classifiers. May obtain better performance but requires greater computational resources[65].

(3) Hybrid methods; this is a combination of both filter and wrapper methods.

(4) Embedded methods; There have been so many useful algorithms that have been developed about feature selection. Among them are;

a) Evolution algorithms (e.g Particle Swarms Optimization, Ant Colony Optimization, etc)

b) Stochastic approaches (e.g Monte Carlo)

c) Genetic algorithms

d) Simulated annealing

With feature selection methods there is no precise which feature selection method is best and can be applied universally, rather a discovery of what can work best for your specific problem needs to be made through series of experimentation.

The figure below shows how to choose a feature selection method[66]

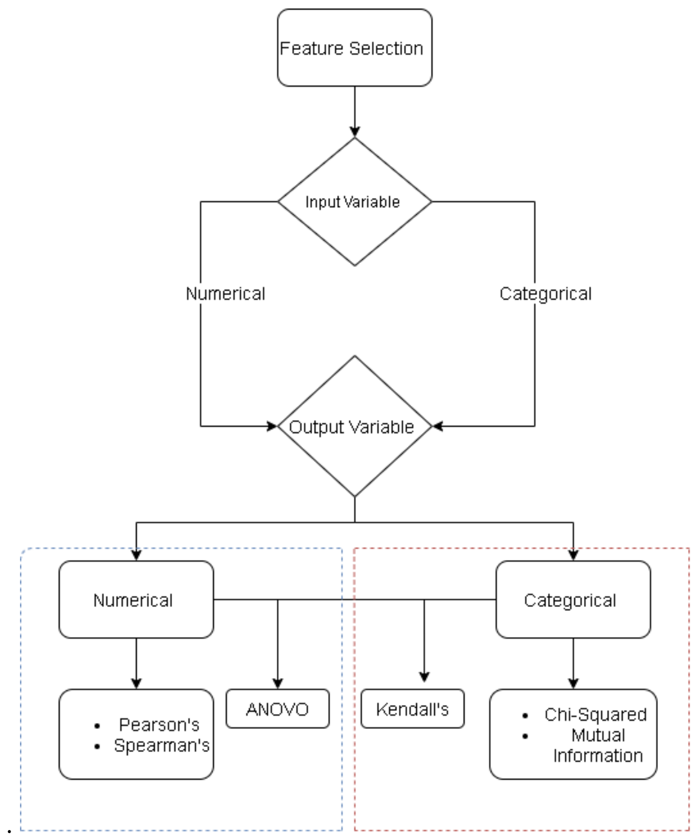


Fig.3-3 Choosing a Feature Selection Method

Step 5: Feature Engineering

This can be described as the process of deriving new variables(useful features) from raw data(available table) using mathematics, statistics, and domain knowledge. This is one of the functions in machine learning which has a huge impact on the performance of a model. Coull et al described Feature engineering tasks in machine learning as the most costly aspect of developing a model, and that cost is even greater in specialized problem domains such as malware classification. It is therefore crucial for both interpretability and performance.

(1) Feature Scaling

Once the features are selected, it is crucial to put the features(variables) on the same scale if possible, as most machine learning algorithms have proven to perform much better when on the same scale. However, although it improves significantly the performance of some algorithms, it does not work well for others because while Gradient Descent Based algorithms (linear and logistic regression and neural network, etc) and Distance-Based Algorithms(KNN, K-means, and SVM) are sensitive to feature scaling, Tree-Based algorithms are insensitive to scale of features. Two main feature scaling techniques are;

a) Normalization(min-max scalling):

Features shifted and rescaled to a range of [0,1] or [-1,1] depending on the nature of the data, which is more of min-max scaling. For each feature column, a min-max formula is applied to it. Below is the general formula for min-max[0,1];

$$X^{'} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad (3-1)$$

In the Formula:

X'—— is the normalized value;

X—— is the original value;

Xmax——are maximum of the features;

Xmin——minimum values of the features.

b) Standardization:

Scaling features based on a normal standard distribution where usually mean = 0 and standard deviation = 1 so that the feature columns have the same parameters as a standard normal distribution.

Below is the formula for standardization;

$$x' = \frac{x - \bar{x}}{\sigma} \qquad (3-2)$$

In the Formula:

x'——is a standardized value;

x——is the original feature vector;

$\bar{x}$——is average (x);

σ——is the standard deviation.

Step 6: Model Building

Next is Model Building, the main goal is to train the best performing model possible, using the preprocessed data. This involves researching the model that will be best for the type of data. In this process, a determination is made whether to use a supervised, unsupervised or reinforced machine learning model. if the data set is labeled, supervised learning can be used, else an unsupervised model is used, reinforced learning is used for the model to decide its next course of action through trial and error to maximize reward. With the supervised model, classification or regression could be used depending on the target variable (Y variable) whether it is categorical or continuous respectively. Furthermore, Unsupervised learning can be used if the input variable (X variable) is unlabeled and uncategorized i.e., the model makes use of only the X variables.

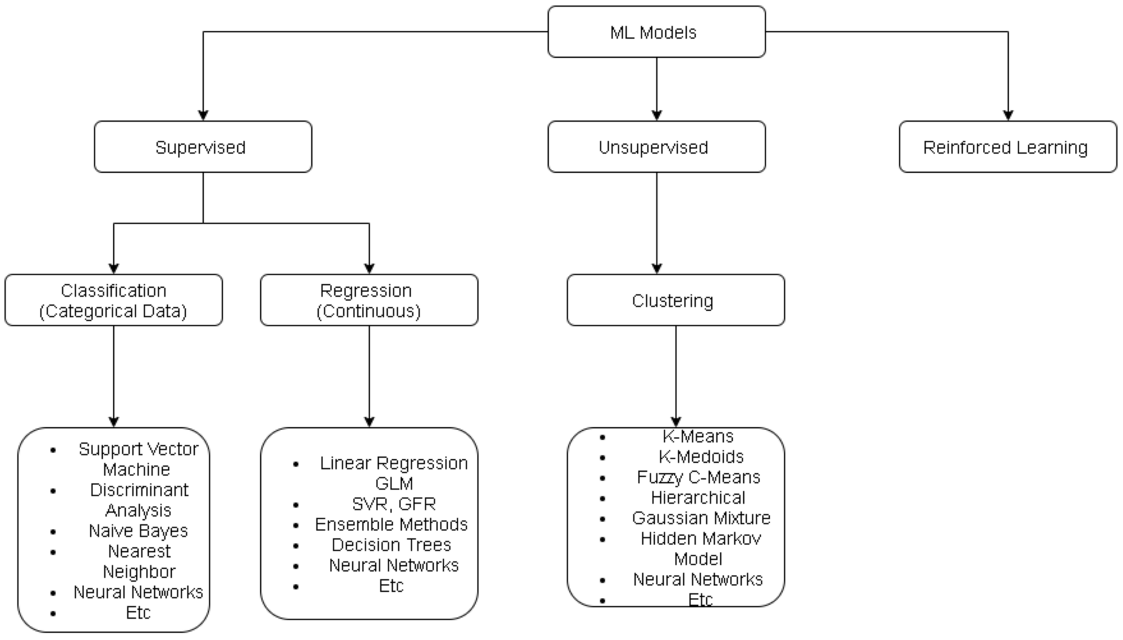


Fig.3-4 Machine Learning Models Overview

(1) Data Splitting - Creating a Train-Test Split

Data splitting is the act of partitioning available data into two portions, usually for cross-validatory purposes. One portion of the data is used to develop a predictive model and the other to evaluate the model's performance[67].Train-Test Split: In this scenario, 80% of the original data is used to train while 20% is used for testing the predictive machine learning model built. The Data split is performed once. Train-Validation-Test Split: data is split into three subsets instead of the two i.e Train set, Validation set, and Testing set. The model is trained with the training set, the validation set is used for the evaluation of the mode while testing of the model is done with the testing set. The ultimate goal of splitting the data is to have the model generalize very well on unseen data after the adjuments of its training and validating parameters.

(2) Overfitting and Underfitting

This is a major problem during the process of building a machine learning model. With underfitting, the model performs poorly on the training data while overfitting occurs when the model performs well on the training data but unable to make a good prediction on new data due to the imbalance between optimization and generalization.

Optimization focus on adjusting the model to create the best model on the training data set during the process of training.

Generalization is keen on how well the model can perform on unseen data. Attaining a model which generalizes best, is the ultimate goal.

At the initial stage of the training, the model tends to underfit since all relevant parameters of the model have not been modeled (learned) yet. However after several iterations, on the training data set, the model will learn the training data set so well that it begins to overfit through the learning of patterns that are too specific to the training data and irrelevant to new data. Limited data availability for the training of the model may lead to overfitting.

(3) Regularization

This is used to prevent overfitting of linear models i.e memorizing patterns instead of generalizing them through penalizing extreme weight values[68]. Also, it can be used when there is no enough data for the training of the model to get it generalized well. Some of the methods of regularization are;

a) Adding weight regularization

b) Reducing the model size

c) L1 regularization; results in sparse models and reduces the amount of noise in the model since it has the effect of reducing the number of features used in the model by pushing to zero the weights of features that will otherwise have small weights.

d) L2 regularization; results in small overall weight values, and stabilizes the weights when there is a high correlation between the input features.

A very large regularization value could result in all features having zero weights, preventing a model from learning patterns.

Hyperparameter Optimization

It can also be described as hyperparameter tuning. Hyperparameters are essential parameters that are used by machine learning algorithms and they directly impact the learning process and prediction performance. Hyperparameters are different from the model's parameters.

Step 7: Model Building

Model Validation is next after building the model with the dataset. In machine learning, model validation is referred to as the process where a trained model is evaluated with a testing data set. The testing data set is a separate portion of the same data set from which the training set is derived. The main purpose of using the testing data set is to test the generalization ability of a trained model[69]. Smith et al indicated that Model validation aims to assess the quality of a model and is possibly the most important step in the model building sequence. Since no method can guarantee the validity of a model with certainty from a finite number of experiments[70]

(1) Cross-Validation

This is a statistical method used to estimate the performance(accuracy) of machine learning models. Amazon ML describes Cross-validation as a technique for evaluating ML models by training several ML models on subsets of the available data and evaluating them on the complementary subset of the data[71]. Non-exhaustive(Holdout, K-fold, stratified K-fold) and exhaustive(Leave-P-Out, Leave-one-out) methods of Cross-Validation can be used. While exhaustive methods test all possible ways to divide the original sample in the training and validation set, Non-exhaustive cross-validation methods do not compute all ways of splitting the original data.

Cross-validation techniques in very useful in determining how good the model performs especially when correcting overfitting of the model. Cross-Validation ensures a more accurate estimate of out-of-sample accuracy and more efficient use of data since every observation is used for both training and testing.

(2) Access Performance

Defining what has been considered a success is key to determining a successful model. Example accuracy, precision, or retention rate.

Poor performance on the training data could be because the model is too simple(input features are not too expressive enough to describe the target well[72]. Making the model flexible could increase the performance through; reducing the amount of regularization, add domain-specific features, and changing the type of feature processing used example increasing n-grams size.

On the other hand, if the model is overfitting the training data, it is important to reduce the model flexibility[72]. The table below shows overfitting and underfitting and the mechanism of correction.

Tab.3-3 Overfitting and Underfitting and their correction mechanism

Problem	Description	Correction Mechanism to Increase Performance
Underfitting	ML model performs poorly on the training data due to the inability to capture the relationship between input and target values	Increase models flexibility by: Decreasing the amount of regularization Add new domain-specific features and change the type of feature processing used (increase n-grams size)
Overfitting	ML model performs well on training data but poorly on the evaluation data due to memorization of the seen data and unable to generalize on unseen data.	Decrease model flexibility by: Increase the amount of regularization Use fewer feature combinations, decrease n-grams size and number of attribute bins.

Step 8: Model Deployment

Deployment of ML model largely depends on the means of the interaction of the model and the end-user. While data scientist is masters of creating models that represent and predict real-world data, the effective deployment of models is best executed by software engineers. However, tools like TFX, MLflow, and Kubeflow can make the process of deployment easy for a data scientist. The difficulty of model deployment and management has created a new specialized role; Machine Learning Engineer, their ideal role is to put ml model into production. The key factors to consider before the commencement of ml deployment include; Data storage and retrieval, Framework and tooling, and Feedback and iteration[73].

3.5 Machine Learning Techniques for Cybersecurity

There are several machine learning algorithms that have been developed by data scientists around the world. However, specific machine learning techniques and algorithms perform and suit well with working on cybersecurity tasks. The table below shows ML techniques in their uses in application to cybersecurity.

Tab.3-4 Machine Learning Techniques and their application in cybersecurity [46]

Technique	Purpose
SVM	1. To classify various attacks such as DoS, Probe, U2R, and R2L 2. Feature selection, intrusion detection, and classification 3. DDoS detection and analysis in SDN-based environment 4. Evaluating host-based anomaly detection systems Bruzzone et al[74], Mohammed et al[75][76], Xiangyan et al[77], Kotpalliwa et al[78], Xie et al[79]
SVM-PSO	1. Detecting and Mitigation of DoS attacks. Divya et al[80] 2. To build an intrusion detection system Ankit et al[81], Harshit et al[82]
FCM clustering, ANN and SVM	1. To build intrusion detection system Chandrasekhar et al[83].
KNN	1. Network intrusion detection system 2. To reduce the false alarm rate Shapoorifard et al[84], Vishwakarma et al[85], Meng et al[86]
SVM and KNN	1. To build intrusion detection system Dada et al[87]
K-means and KNN	1. To build intrusion detection system. Li et al[88]
KNN and Clustering	1. To build intrusion detection system. Lin et al [89]
Naive Bayes	1. To build an intrusion detection system for multi-class classification. Koc et al[90]
Decision Tree	1. For Feature selection, and to build an effective network intrusion detection system. Moon et al[91], Ingre et al[92], Malik et al[93] , Relan et al[94], Rai et al[95], Sarker et al[91], Puthran et al[96]
Decision Tree and KNN	1. Anomaly intrusion detection system. Balogun et al[97]
Genetic Algorithm and Decision Tree	1. To solve the problem of small disjunct in the decision tree-based intrusion detection system. Azad et al[84]
Decision Tree and ANN	1. To measure the performance of intrusion detection systems. Ullah et al[98]
Random Forests	1. To build network intrusion detection systems. Zhang et al[88]

表 3-4（续）

Technique	Purpose
Association Rule	1. To build network intrusion detection systems. Tajbakhsh et al[99]
Behavior Rule	1. To build intrusion detection system for safety-critical medical cyber-physical systems. Mitchell et al[100]
Supervised	1. For malware detection and analysis. Alazab et al[101]
Semi-supervised Adaboost	1. For network anomaly detection. Yuan et al[102]
Hidden Markov Models	1. To build an intrusion detection system. Ariu et al[102], Aarnes et al[43].
Genetic Algorithm	1. For prevention of cyberterrorism through dynamic and evolving intrusion detection. Hansen et al[36], Aslahi et al[103]
Deep Learning Recurrent, RNN, LSTM	1. To build anomaly intrusion detection system and attack classification. Yin et al[37] , Kim et al[104], Almiani et al[38]
Deep Learning Convolutional	1. Malware traffic classifcation system. Kolosnjaji et al[105]
Deep and Reinforcement Learning	1. Malicious activities and intrusion detection system. Alauthman et al[106], Blanco et al[107], Lopez et al[108]

3.6 Machine Learning Vs Deep Learning Vs Neural Deep Learning

Since deep learning and machine learning tend to be used interchangeably, it’s worth noting the nuances between the two. Machine learning, deep learning, and neural networks are all sub-fields of artificial intelligence. However, deep learning is a sub-field of machine learning, and neural networks are a sub-field of deep learning. The way in which deep learning and machine learning differ is in how each algorithm learns. Deep learning automates much of the feature extraction piece of the process, eliminating some of the manual human intervention required and enabling the use of larger data sets. You can think of deep learning as "scalable machine learning". Classical, or "non-deep", machine learning is more dependent on human intervention to learn. Human experts determine the set of features to understand the differences between data inputs, usually requiring more structured data to learn. "Deep" machine learning can leverage labeled datasets, also known as supervised learning, to inform its algorithm, but it doesn’t necessarily require a labeled dataset. Neural networks, or artificial neural networks (ANNs), are comprised of a node layer, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold[43].

3.7 Summary

This chapter gives an indebt description of the elements of malware detection and its application in machine learning are discussed.

4 EXPERIMENTATIONS

In this section, we undertake series of activities to demonstrate the processes involved in the implementation of supervised machine learning for malware detection. We started from the very beginning of all the relevant processes of raw data collection and processing and then narrowed down the implementation of the framework of machine learning for malware detection. We used our extracted samples to build our model.

4.1 Overview

Our experiment was performed on samples of executable files obtain from open-source data platforms; GitHub and Kaggle. For the purposes of our academic work, we obtained our malware sample from open source GitHub ytisf/theZoo[109] and Kaggle Microsoft Malware Classification Challenge (BIG 2015)[110]. These data sources are all public free data and can be accessed through the respective websites which source links have been given in our references.

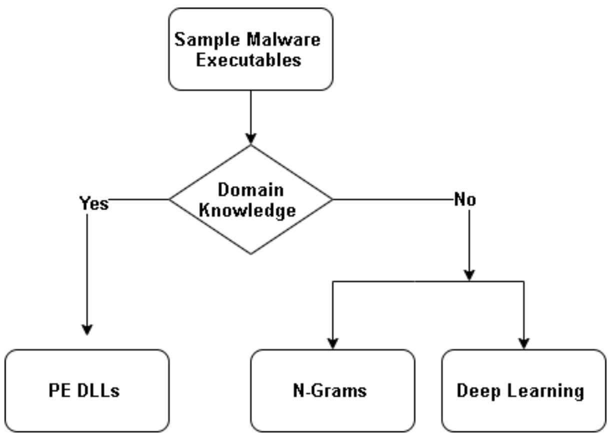


Fig.4-1 General Experiment overview

From the figure above, PE DLLs are used to demonstrate a situation where there exists domain knowledge of the samples while N-Grams and Deep Learning are deployed in a situation where there is no domain knowledge of the available sample files.

4.1.1 Supervised Learning (Domain Knowledge)

As already indicated in chapter three, Supervised learning is performed when specific targets are defined to reach from a certain set of inputs, i.e., task-driven approach. In the area of machine learning, the most popular supervised learning techniques are known as classification and regression methods. These techniques are popular to classify or predict the future of a particular security problem. For instance, to predict denial-of-service attacks (yes, no) or to identify different classes of network attacks such as scanning and spoofing, classification techniques can be used in the cybersecurity domain[43]. In our experiment, we predicted malware using the PE DLL information to classify Malicious and Benign Samples.

4.1.2 Unsupervised Learning (No Domain Knowledge)

In unsupervised learning problems, the main task is to find patterns, structures, or knowledge in unlabeled data, i.e., a data-driven approach. In the area of cybersecurity, cyber-attacks like malware stays hidden in some ways, including changing their behavior dynamically and autonomously to avoid detection. Clustering techniques, a type of unsupervised learning, can help to uncover the hidden patterns and structures from the datasets, to identify indicators of such sophisticated attacks. Similarly, in identifying anomalies, policy violations, detecting, and eliminating noisy instances in data, clustering techniques can be useful. K-means, K-medoids are the popular partitioning clustering algorithms, and single linkage or complete linkage are the well-known hierarchical clustering algorithms used in various application domains. Moreover, a bottom-up clustering approach proposed by Sarker et al. can also be used by taking into account the data characteristics[51].

4.2 Preparation of Samples

This is the initial part of our experiment where we prepare our samples and generate corpus from the available sources. The following steps were undertaken:

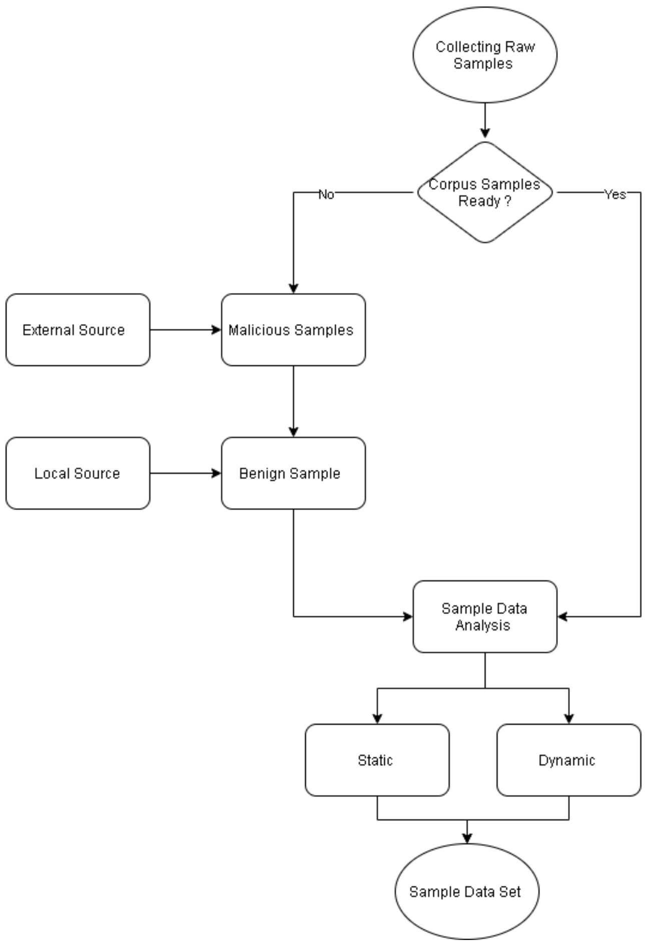
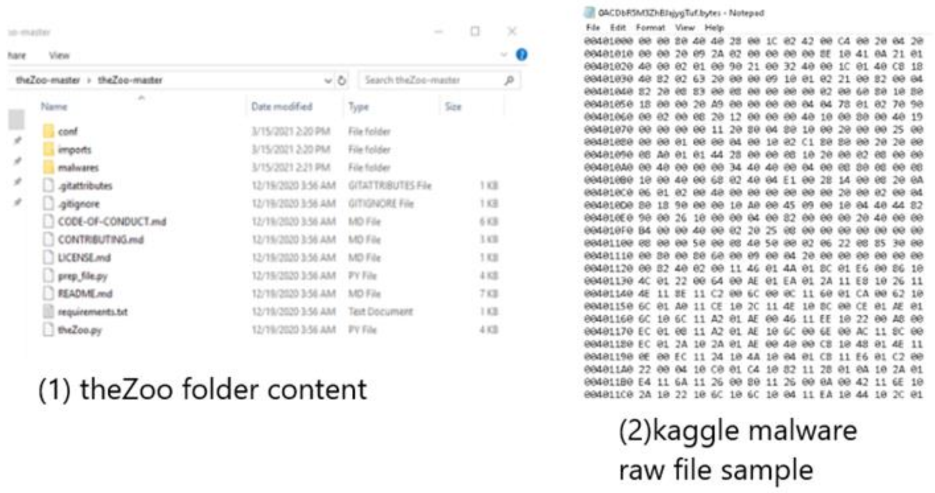


Fig.4-2 Security Data Samples set preparation process

Step 1: Collection of Security Data

The malware from the Zoo is a Live malware i.e can be executed because it has the PE headers, the malware on therefore extra caution needed to be taken as has execution capabilities and can be activated at any instance. The malware from Kaggle Microsoft is Not Live i.e cannot be executed because has no PE headers.



theZoo-master › theZoo-master

Name	Date modified	Type	Size
conf	3/15/2021 2:20 PM	File folder	
imports	3/15/2021 2:20 PM	File folder	
malwares	3/15/2021 2:21 PM	File folder	
.gitattributes	12/19/2020 3:56 AM	GITATTRIBUTES File	1 KB
.gitignore	12/19/2020 3:56 AM	GITIGNORE File	1 KB
CODE-OF-CONDUCT.md	12/19/2020 3:56 AM	MD File	6 KB
CONTRIBUTING.md	12/19/2020 3:56 AM	MD File	3 KB
LICENSE.md	12/19/2020 3:56 AM	MD File	1 KB
prep_file.py	12/19/2020 3:56 AM	PY File	4 KB
README.md	12/19/2020 3:56 AM	MD File	7 KB
requirements.txt	12/19/2020 3:56 AM	Text Document	1 KB
theZoo.py	12/19/2020 3:56 AM	PY File	4 KB

0ACDbR5M3ZhBJajygTuf.bytes - Notepad

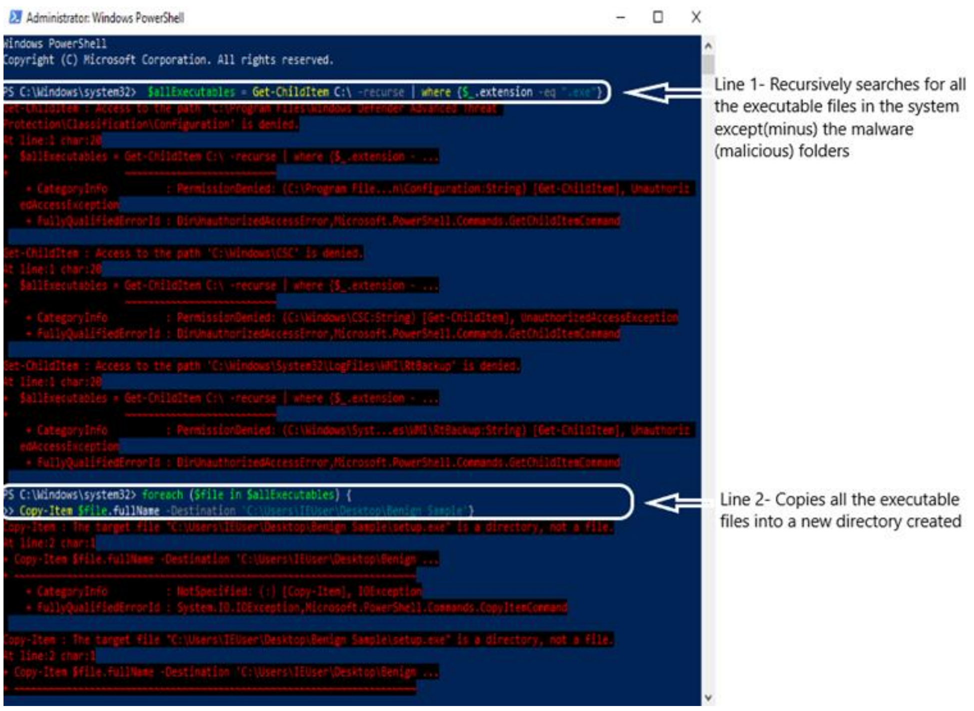
```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 00 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
00401060 00 02 00 08 20 12 00 00 00 40 10 00 80 00 40 19
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00
00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00
004010A0 00 40 00 00 00 34 40 40 00 04 00 08 80 08 00 08
004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A
004010C0 06 01 02 00 40 00 00 00 00 00 00 20 00 02 00 04
004010D0 80 18 90 00 00 10 A0 00 45 09 00 10 04 40 44 82
004010E0 90 00 26 10 00 00 04 00 82 00 00 00 20 40 00 00
004010F0 B4 00 00 40 00 02 20 25 08 00 00 00 00 00 00 00
00401100 08 00 00 50 00 08 40 50 00 02 06 22 08 85 30 00
00401110 00 B0 00 80 60 00 09 00 04 20 00 00 00 00 00 00
00401120 00 82 40 02 00 11 46 01 4A 01 BC 01 E6 00 86 10
00401130 4C 01 22 00 64 00 AE 01 EA 01 2A 11 E8 10 26 11
00401140 4E 11 8E 11 C2 00 6C 00 0C 11 60 01 CA 00 62 10
00401150 6C 01 A0 11 CE 10 2C 11 4E 10 BC 00 CE 01 AE 01
00401160 6C 10 6C 11 A2 01 AE 00 46 11 EE 10 22 00 A8 00
00401170 EC 01 08 11 A2 01 AE 10 6C 00 6E 00 AC 11 8C 00
00401180 EC 01 2A 10 2A 01 AE 00 40 00 C8 10 48 01 4E 11
00401190 0E 00 EC 11 24 10 4A 10 04 01 C8 11 E6 01 C2 00
004011A0 22 00 04 10 C0 01 C4 10 82 11 28 01 0A 10 2A 01
004011B0 E4 11 6A 11 26 00 80 11 26 00 0A 00 42 11 6E 10
004011C0 2A 10 22 10 6C 10 6C 10 04 11 EA 10 44 10 2C 01
```

Fig.4-3 theZoo folder content(1) and Kaggle MS BIG(2) malware raw sample

Figure 4-3 shows the folder content of theZoo malware which contains a significant amount of live malware and a data file sample of the Kaggle Microsoft malware of which for each file, the raw data contains the hexadecimal representation of the file's binary content, without the PE header.

Step 2: Obtaining the Benign Data Set

We used a power shell to obtain the benign data set so that having both the benign and malicious data sets, a classifier can be created to be able to determine if an unknown sample is benign or malicious. One of the ways we use to get the benign data set is to search for all executable files on the system except for the malware files (theZoo files) in the system, all other files will be benign. The figure below shows benign files being extracted to a new directory using a power shell.


Administrator: Windows PowerShell
Line 1- Recursively searches for all the executable files in the system except(minus) the malware (malicious) folders
Line 2- Copies all the executable files into a new directory created

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> $allExecutables = Get-ChildItem C:\ -recurse | where {$_.extension -eq ".exe"}
Get-ChildItem : Access to the path 'C:\Program Files\Windows Defender Advanced Threat
Protection\Classification\Configuration' is denied.
At line:1 char:20
+ $allExecutables = Get-ChildItem C:\ -recurse | where {$_.extension - ...
+                   ~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (C:\Program File...n\Configuration:String) [Get-ChildItem], Unauthoriz
   edAccessException
    + FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand

Get-ChildItem : Access to the path 'C:\Windows\CSC' is denied.
At line:1 char:20
+ $allExecutables = Get-ChildItem C:\ -recurse | where {$_.extension - ...
+                   ~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (C:\Windows\CSC:String) [Get-ChildItem], UnauthorizedAccessException
    + FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand

Get-ChildItem : Access to the path 'C:\Windows\System32\LogFiles\WMI\RtBackup' is denied.
At line:1 char:20
+ $allExecutables = Get-ChildItem C:\ -recurse | where {$_.extension - ...
+                   ~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (C:\Windows\Syst...es\WMI\RtBackup:String) [Get-ChildItem], Unauthoriz
   edAccessException
    + FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand

PS C:\Windows\system32> foreach ($file in $allExecutables) {
>> Copy-Item $file.fullName -Destination 'C:\Users\IEUser\Desktop\Benign Sample'}
Copy-Item : The target file "C:\Users\IEUser\Desktop\Benign Sample\setup.exe" is a directory, not a file.
At line:2 char:1
+ Copy-Item $file.fullName -Destination 'C:\Users\IEUser\Desktop\Benign ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [Copy-Item], IOException
    + FullyQualifiedErrorId : System.IO.IOException,Microsoft.PowerShell.Commands.CopyItemCommand

Copy-Item : The target file "C:\Users\IEUser\Desktop\Benign Sample\setup.exe" is a directory, not a file.
At line:2 char:1
+ Copy-Item $file.fullName -Destination 'C:\Users\IEUser\Desktop\Benign ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

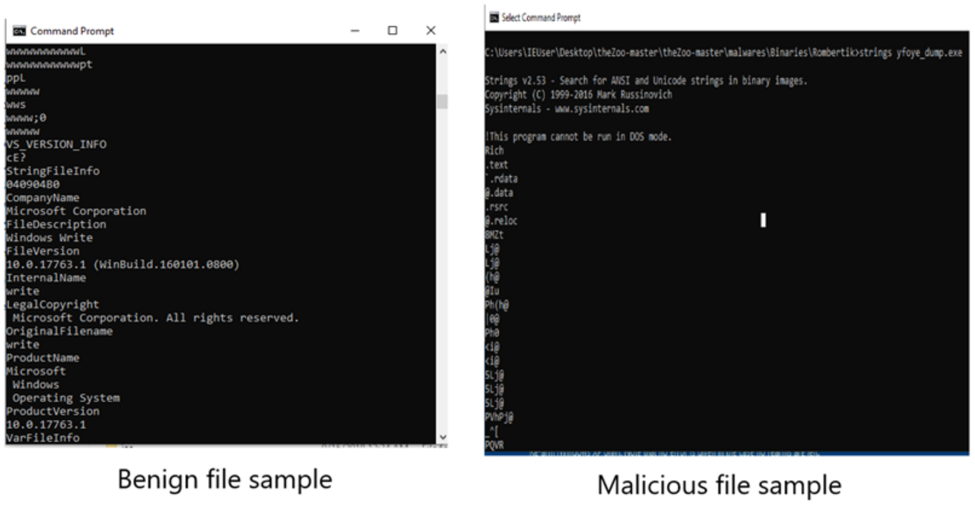
Fig.4-4 Extracting Benign Files from the Windows system using Powershell

Step 3: Data Preparation

(1) Static Analysis

As already discussed in chapter 2, with static analysis, analysis is performed on files without executing those files.

Strings: For the static analysis we used the Microsoft tool Strings[111] to scan our benign and malicious files for their UNICODE or ASCII strings. This reveals their characteristics and through accessing the characteristics a preliminary determination can be made of the file as to whether they are benign or malicious. The figure below shows the string tool passed on Benign and malicious files respectively. However, this process can be biased as some information on the file can be hidden through the obfuscation process.



```
wwwwwwwwwwwL
wwwwwwwwwwwpt
ppL
wwwww
wws
wwww;0
wwwww
VS_VERSION_INFO
cE?
StringFileInfo
040904B0
CompanyName
Microsoft Corporation
FileDescription
Windows Write
FileVersion
10.0.17763.1 (WinBuild.160101.0800)
InternalName
write
LegalCopyright
 Microsoft Corporation. All rights reserved.
OriginalFilename
write
ProductName
Microsoft
 Windows
 Operating System
ProductVersion
10.0.17763.1
VarFileInfo
```

```
C:\Users\IEUser\Desktop\theZoo-master\theZoo-master\malwares\Binaries\Rombertik>strings yfoye_dump.exe

Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.
Rich
.text
`.rdata
@.data
.rsrc
@.reloc
8MZt
Lj@
Lj@
(h@
@Iu
Ph(h@
|@@
Ph@
<i@
<i@
5Lj@
5Lj@
5Lj@
PVhPj@
_^[
PQVR
```

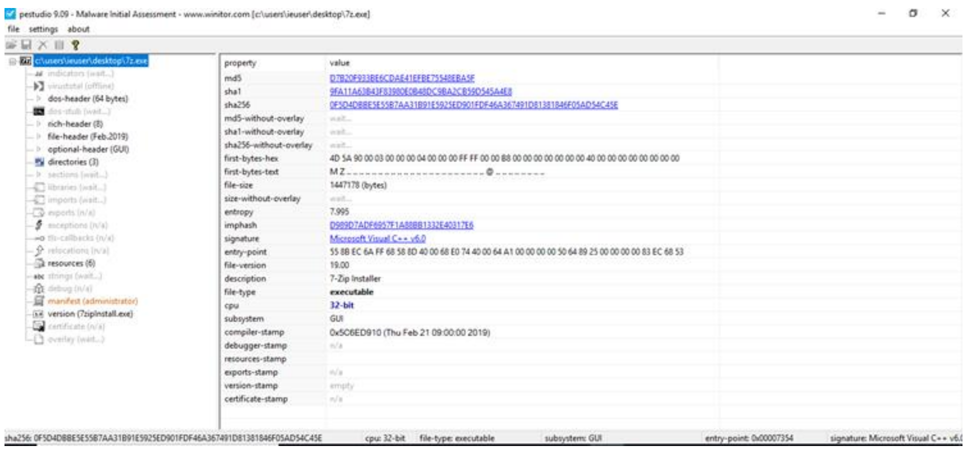
Fig.4-5 String Passed on Benign and Malicious Files

(2) Dynamic Analysis

a) Dynamic analysis with PE studio

As discussed already in chapter 3, dynamic analysis requires files to be executed and then analyzed. Although requires extra care as the system can be affected by the malware, it is more effective and efficient in unveiling malware hidden in a file than static analysis.

PE Studio: We used PE studio to perform dynamic analysis. It reveals more information about the file PE headers, Libraries, drivers, etc. Below figure 11 shows a benign file sample run in PE studio.



pestudio 9.09 - Malware Initial Assessment - www.winitor.com [c:\users\ieuser\desktop\7z.exe]

file settings about

c:\users\ieuser\desktop\7z.exe
- indicators (wait...)
- virustotal (offline)
- dos-header (64 bytes)
- dos-stub (wait...)
- rich-header (8)
- file-header (Feb.2019)
- optional-header (GUI)
- directories (3)
- sections (wait...)
- libraries (wait...)
- imports (wait...)
- exports (n/a)
- exceptions (n/a)
- tls-callbacks (n/a)
- relocations (n/a)
- resources (6)
- strings (wait...)
- debug (n/a)
- manifest (administrator)
- version (7zipInstall.exe)
- certificate (n/a)
- overlay (wait...)

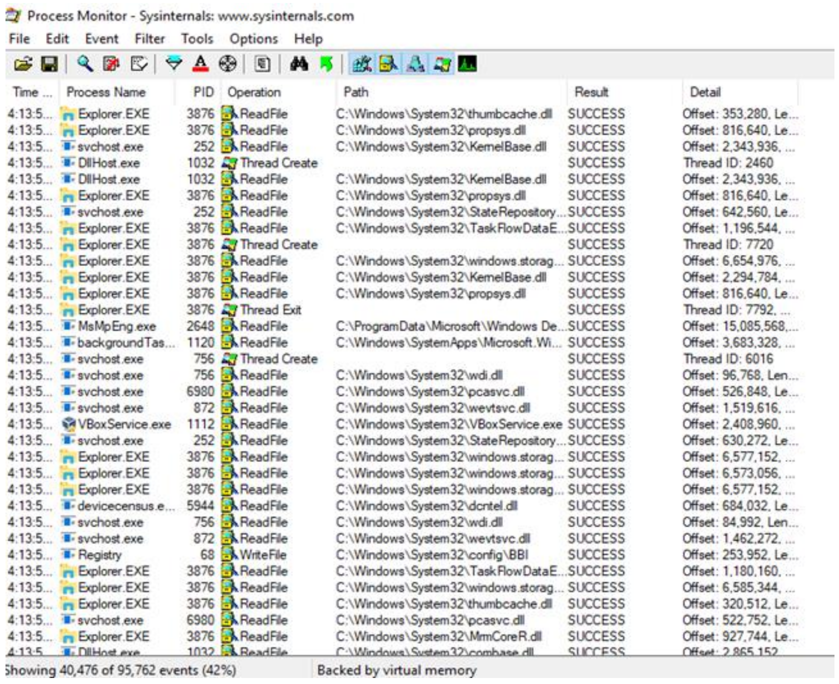
property	value
md5	D7B20F933BE6CDAE41EFBE75548EBA5F
sha1	9FA11A63B43F83980E0B48DC9BA2CB59D545A4E8
sha256	0F5D4DBBE5E55B7AA31B91E5925ED901FDF46A367491D81381846F05AD54C45E
md5-without-overlay	wait...
sha1-without-overlay	wait...
sha256-without-overlay	wait...
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00
first-bytes-text	M Z _ @ _ _ _ _ _ _ _ _
file-size	1447178 (bytes)
size-without-overlay	wait...
entropy	7.995
imphash	D989D7ADF6957F1A88BB1332E40317E6
signature	Microsoft Visual C++ v6.0
entry-point	55 8B EC 6A FF 68 58 8D 40 00 68 E0 74 40 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 EC 68 53
file-version	19.00
description	7-Zip Installer
file-type	**executable**
cpu	**32-bit**
subsystem	GUI
compiler-stamp	0x5C6ED910 (Thu Feb 21 09:00:00 2019)
debugger-stamp	n/a
resources-stamp	
exports-stamp	n/a
version-stamp	empty
certificate-stamp	n/a

sha256: 0F5D4DBBE5E55B7AA31B91E5925ED901FDF46A367491D81381846F05AD54C45E cpu: 32-bit file-type: executable subsystem: GUI entry-point: 0x00007354 signature: Microsoft Visual C++ v6.0

Fig.4-6 PE Studio running a Benign file

b) Dynamic Analysis Windows Process Monitor

We used Windows Process Monitor for the performance of our dynamic analysis. We captured all the activities going on in the system. While the malicious file was running, activities such as changing the system registry, creating files and folders as well as encryption of folders and files could be observed. The figure below shows a process monitor capturing system activities.



Process Monitor - Sysinternals: www.sysinternals.com

Time ...	Process Name	PID	Operation	Path	Result	Detail
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\thumbcache.dll	SUCCESS	Offset: 353,280, Le...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\propsys.dll	SUCCESS	Offset: 816,640, Le...
4:13:5...	svchost.exe	252	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 2,343,936, ...
4:13:5...	DllHost.exe	1032	Thread Create		SUCCESS	Thread ID: 2460
4:13:5...	DllHost.exe	1032	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 2,343,936, ...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\propsys.dll	SUCCESS	Offset: 816,640, Le...
4:13:5...	svchost.exe	252	ReadFile	C:\Windows\System32\StateRepository...	SUCCESS	Offset: 642,560, Le...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\TaskFlowDataE...	SUCCESS	Offset: 1,196,544, ...
4:13:5...	Explorer.EXE	3876	Thread Create		SUCCESS	Thread ID: 7720
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\windows.storag...	SUCCESS	Offset: 6,654,976, ...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\KernelBase.dll	SUCCESS	Offset: 2,294,784, ...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\propsys.dll	SUCCESS	Offset: 816,640, Le...
4:13:5...	Explorer.EXE	3876	Thread Exit		SUCCESS	Thread ID: 7792, ...
4:13:5...	MsMpEng.exe	2648	ReadFile	C:\ProgramData\Microsoft\Windows De...	SUCCESS	Offset: 15,085,568,...
4:13:5...	backgroundTas...	1120	ReadFile	C:\Windows\SystemApps\Microsoft.Wi...	SUCCESS	Offset: 3,683,328, ...
4:13:5...	svchost.exe	756	Thread Create		SUCCESS	Thread ID: 6016
4:13:5...	svchost.exe	756	ReadFile	C:\Windows\System32\wdi.dll	SUCCESS	Offset: 96,768, Len...
4:13:5...	svchost.exe	6980	ReadFile	C:\Windows\System32\pcasvc.dll	SUCCESS	Offset: 526,848, Le...
4:13:5...	svchost.exe	872	ReadFile	C:\Windows\System32\wevtsvc.dll	SUCCESS	Offset: 1,519,616, ...
4:13:5...	VBoxService.exe	1112	ReadFile	C:\Windows\System32\VBoxService.exe	SUCCESS	Offset: 2,408,960, ...
4:13:5...	svchost.exe	252	ReadFile	C:\Windows\System32\StateRepository...	SUCCESS	Offset: 630,272, Le...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\windows.storag...	SUCCESS	Offset: 6,577,152, ...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\windows.storag...	SUCCESS	Offset: 6,573,056, ...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\windows.storag...	SUCCESS	Offset: 6,577,152, ...
4:13:5...	devicecensus.e...	5944	ReadFile	C:\Windows\System32\dcntel.dll	SUCCESS	Offset: 684,032, Le...
4:13:5...	svchost.exe	756	ReadFile	C:\Windows\System32\wdi.dll	SUCCESS	Offset: 84,992, Len...
4:13:5...	svchost.exe	872	ReadFile	C:\Windows\System32\wevtsvc.dll	SUCCESS	Offset: 1,462,272, ...
4:13:5...	Registry	68	WriteFile	C:\Windows\System32\config\BBI	SUCCESS	Offset: 253,952, Le...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\TaskFlowDataE...	SUCCESS	Offset: 1,180,160, ...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\windows.storag...	SUCCESS	Offset: 6,585,344, ...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\thumbcache.dll	SUCCESS	Offset: 320,512, Le...
4:13:5...	svchost.exe	6980	ReadFile	C:\Windows\System32\pcasvc.dll	SUCCESS	Offset: 522,752, Le...
4:13:5...	Explorer.EXE	3876	ReadFile	C:\Windows\System32\MrmCoreR.dll	SUCCESS	Offset: 927,744, Le...
4:13:5...	DllHost.exe	1032	ReadFile	C:\Windows\System32\combase.dll	SUCCESS	Offset: 2,865,152

Showing 40,476 of 95,762 events (42%) Backed by virtual memory

Fig.4-7 Process Monitor capturing system activities

4.3 PE File DLL to Classify Malware Implementation

In this section of our experiment, we used the PE file DLL information to classify samples as benign or malicious files. Once the data preparation is completed and all the sample and corpus are ready from the previous data preparation stage, the following steps were undertaken:

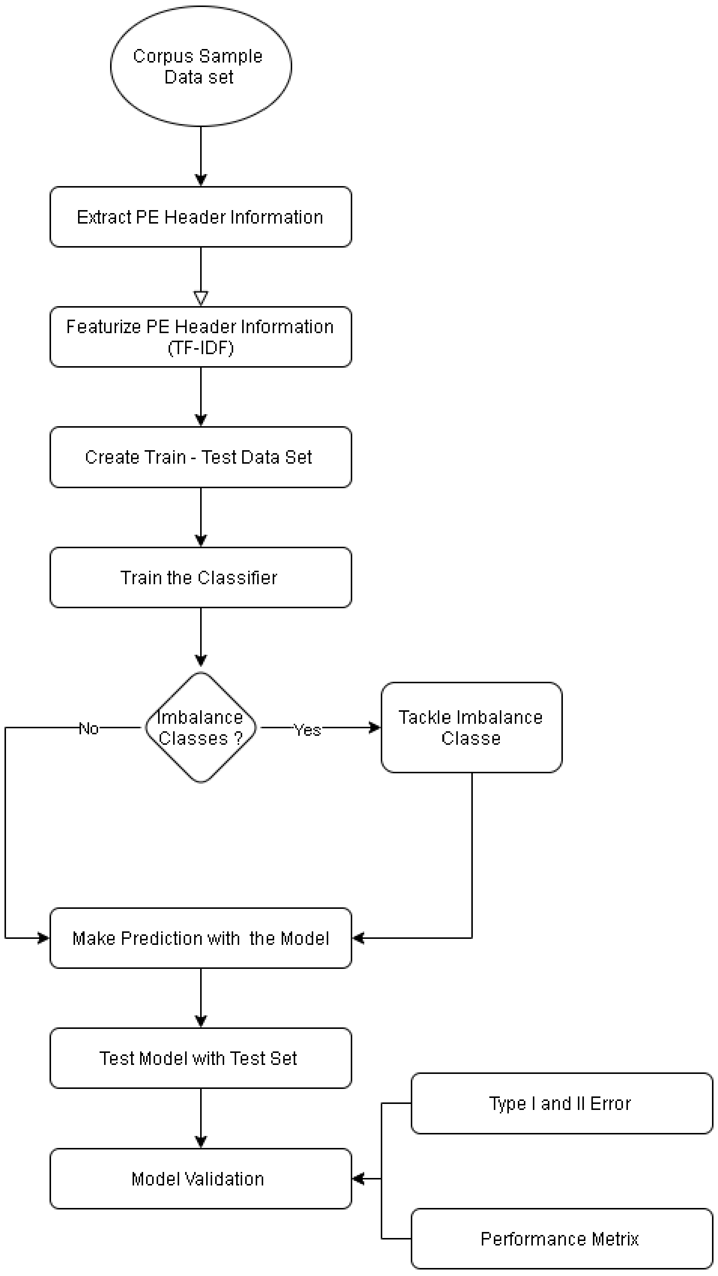


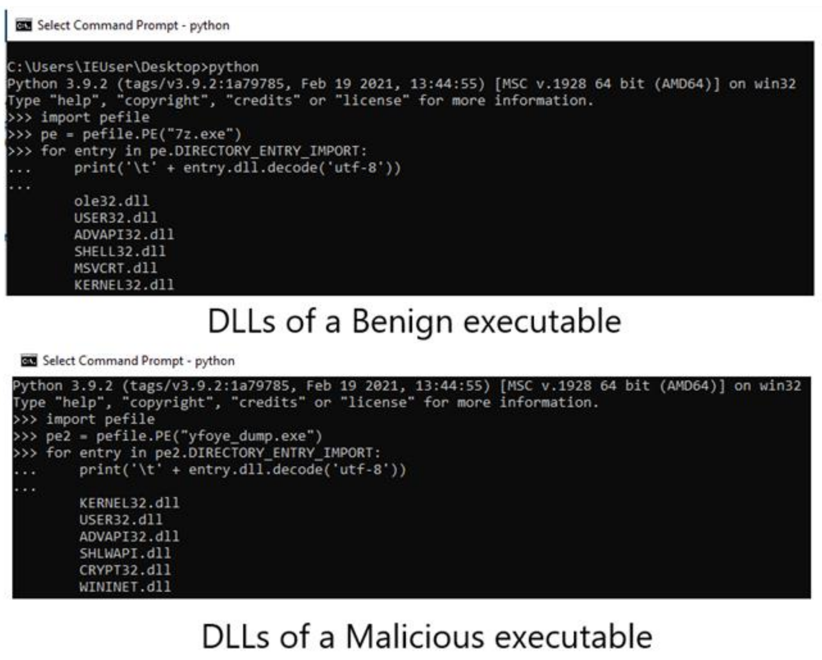
Fig.4-8 Implementation of PE DLLs for Malware Detection Architecture

Step 1: Feature Selection

a) Extracting PE information from PE File

We used the python library PE file to extract relevant features from the PE file which were then used for the building of our classifiers. With domain knowledge, a determination is made that imported DLLs found in the Import Address Table are the most important features that distinguish between benign and malicious samples. For example, is when an execution file sample is importing a crypto library then it is highly likely to be identified as ransomware.

The figure below shows a sample of Bening and Malicious DLLs imported by an executable.



Select Command Prompt - python

```
C:\Users\IEUser\Desktop>python
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pefile
>>> pe = pefile.PE("7z.exe")
>>> for entry in pe.DIRECTORY_ENTRY_IMPORT:
...     print('\t' + entry.dll.decode('utf-8'))
...
        ole32.dll
        USER32.dll
        ADVAPI32.dll
        SHELL32.dll
        MSVCRT.dll
        KERNEL32.dll
```

DLLs of a Benign executable
Select Command Prompt - python

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pefile
>>> pe2 = pefile.PE("yfoye_dump.exe")
>>> for entry in pe2.DIRECTORY_ENTRY_IMPORT:
...     print('\t' + entry.dll.decode('utf-8'))
...
        KERNEL32.dll
        USER32.dll
        ADVAPI32.dll
        SHLWAPI.dll
        CRYPT32.dll
        WININET.dll
```

Fig.4-9 Imported DLLs of a Malicious and Benign Executable

From the figure above, in the malicious executable, an Import call has been made for a Crypto library. This makes the malicious file more likely to be crypto-malware. In examining the PE file, we first looked at the models it offers and then selected our desired option which Directory Entry import.

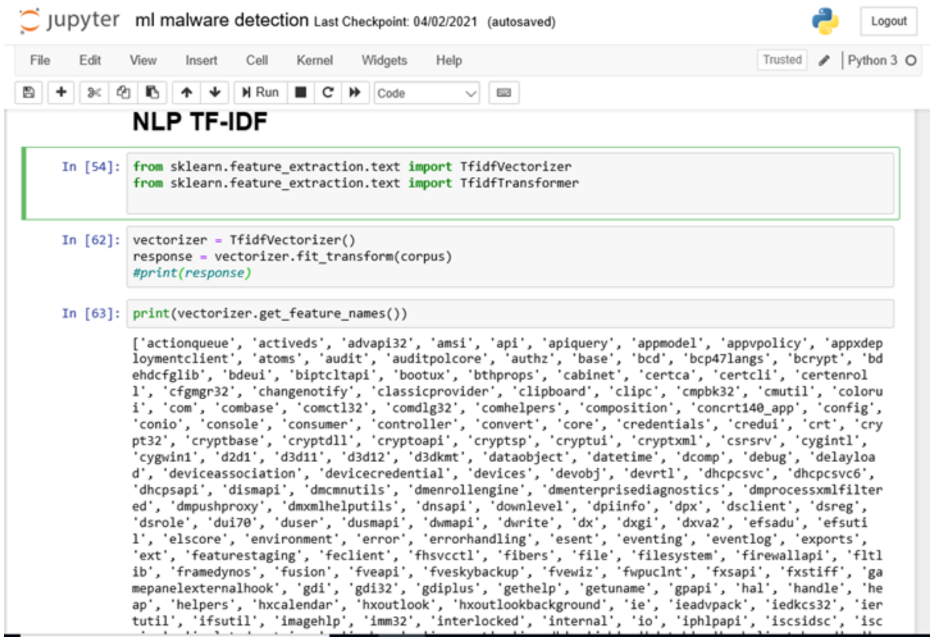
b) Futurizing The PE Header Information Using TF-IDF

Term Frequency- Inverse Document Frequency as the term implies, TF-IDF calculates values for each word in a document through an inverse proportion of the frequency of the word in a particular document to the percentage of documents the word appears in. Words with high TF-IDF numbers imply a strong relationship with the document they appear in, suggesting that if that word were to appear in a query, the document could be of interest to the user [112].

We used the TF-IDF technique in sklearn to convert the words in the list of DLLs into vectors. Thus, counting the frequency of each word per document. This was achieved by counting the frequency of each word per document. Large weights are given to significant words that appear less frequently whiles smaller weights are assigned to words the appear frequently but are less significant.

The goal is to create a classifier that will read in the list of DLLs and then use it to predict whether a sample is benign or malicious. While TF records the number of occurrences of a word, IDF attaches significance to words by given small weights to common words and larger weights to important words.

The figure below shows features from DLLs extracted using sklearn.


jupyter ml malware detection Last Checkpoint: 04/02/2021 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

NLP TF-IDF

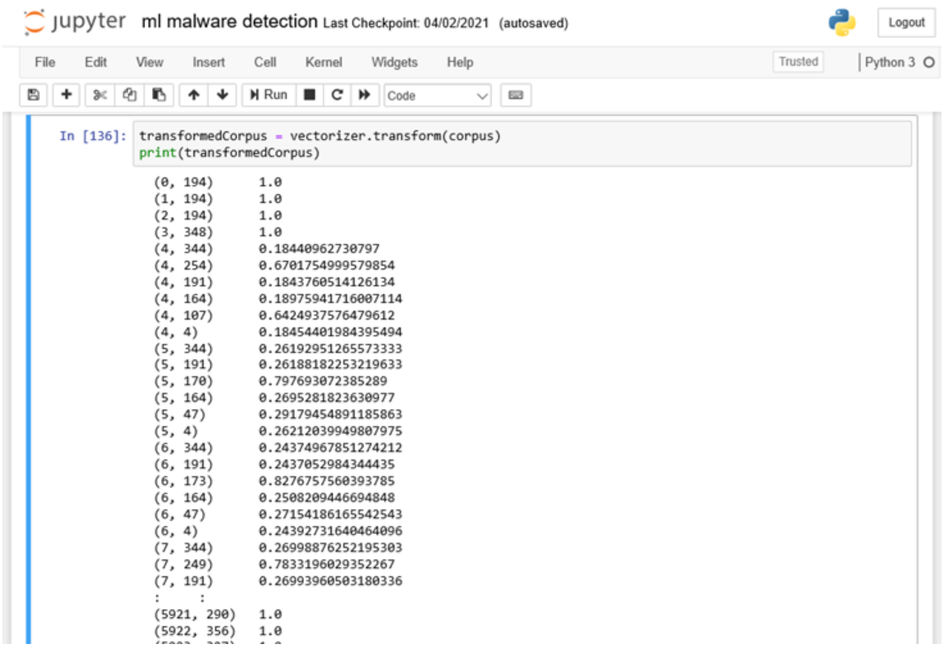
```
In [54]: from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn.feature_extraction.text import TfidfTransformer

In [62]: vectorizer = TfidfVectorizer()
         response = vectorizer.fit_transform(corpus)
         #print(response)

In [63]: print(vectorizer.get_feature_names())

         ['actionqueue', 'activeds', 'advapi32', 'amsi', 'api', 'apiquery', 'appmodel', 'appvpolicy', 'appxdep
         loymentclient', 'atoms', 'audit', 'auditpolcore', 'authz', 'base', 'bcd', 'bcp47langs', 'bcrypt', 'bd
         ehdcfglib', 'bdeui', 'biptcltapi', 'bootux', 'bthprops', 'cabinet', 'certca', 'certcli', 'certenrol
         l', 'cfgmgr32', 'changenotify', 'classicprovider', 'clipboard', 'clipc', 'cmpbk32', 'cmutil', 'coloru
         i', 'com', 'combase', 'comctl32', 'comdlg32', 'comhelpers', 'composition', 'concrt140_app', 'config',
         'conio', 'console', 'consumer', 'controller', 'convert', 'core', 'credentials', 'credui', 'crt', 'cry
         pt32', 'cryptbase', 'cryptdll', 'cryptoapi', 'cryptsp', 'cryptui', 'cryptxml', 'csrsrv', 'cygintl',
         'cygwin1', 'd2d1', 'd3d11', 'd3d12', 'd3dkmt', 'dataobject', 'datetime', 'dcomp', 'debug', 'delayloa
         d', 'deviceassociation', 'devicecredential', 'devices', 'devobj', 'devrtl', 'dhcpcsvc', 'dhcpcsvc6',
         'dhcpsapi', 'dismapi', 'dmcmnutils', 'dmenrollengine', 'dmenterprisediagnostics', 'dmprocessxmlfilter
         ed', 'dmpushproxy', 'dmxmlhelputils', 'dnsapi', 'downlevel', 'dpiinfo', 'dpx', 'dsclient', 'dsreg',
         'dsrole', 'dui70', 'duser', 'dusmapi', 'dwmapi', 'dwrite', 'dx', 'dxgi', 'dxva2', 'efsadu', 'efsuti
         l', 'elscore', 'environment', 'error', 'errorhandling', 'esent', 'eventing', 'eventlog', 'exports',
         'ext', 'featurestaging', 'feclient', 'fhsvcctl', 'fibers', 'file', 'filesystem', 'firewallapi', 'fltl
         ib', 'framedynos', 'fusion', 'fveapi', 'fveskybackup', 'fvewiz', 'fwpuclnt', 'fxsapi', 'fxstiff', 'ga
         mepanelexternalhook', 'gdi', 'gdi32', 'gdiplus', 'gethelp', 'getuname', 'gpapi', 'hal', 'handle', 'he
         ap', 'helpers', 'hxcalendar', 'hxoutlook', 'hxoutlookbackground', 'ie', 'ieadvpack', 'iedkcs32', 'ier
         tutil', 'ifsutil', 'imagehlp', 'imm32', 'interlocked', 'internal', 'io', 'iphlpapi', 'iscsidsc', 'isc
```

Fig.4-10 sklearn used to extract features from DLLs


jupyter ml malware detection Last Checkpoint: 04/02/2021 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [136]: transformedCorpus = vectorizer.transform(corpus)
          print(transformedCorpus)

            (0, 194)      1.0
            (1, 194)      1.0
            (2, 194)      1.0
            (3, 348)      1.0
            (4, 344)      0.18440962730797
            (4, 254)      0.6701754999579854
            (4, 191)      0.1843760514126134
            (4, 164)      0.18975941716007114
            (4, 107)      0.6424937576479612
            (4, 4)        0.18454401984395494
            (5, 344)      0.26192951265573333
            (5, 191)      0.26188182253219633
            (5, 170)      0.797693072385289
            (5, 164)      0.2695281823630977
            (5, 47)       0.29179454891185863
            (5, 4)        0.26212039949807975
            (6, 344)      0.24374967851274212
            (6, 191)      0.2437052984344435
            (6, 173)      0.8276757560393785
            (6, 164)      0.2508209446694848
            (6, 47)       0.27154186165542543
            (6, 4)        0.24392731640464096
            (7, 344)      0.26998876252195303
            (7, 249)      0.7833196029352267
            (7, 191)      0.26993960503180336
            :     :
            (5921, 290)   1.0
            (5922, 356)   1.0
```

Fig.4-11 Converting Text to figures and vectorizing the figures.

Step 2: Feature Engineering

For our feature engineering, we prepared a corpus which consists of an import address table of each sample.

Using python and jupyter notebook we iterated through all the files of our Benign and Malicious samples to extract the features from the PE information of the executable files. All the import address table (pe DIRECTORY_ENTRY_IMPORT) of each sample of the executables was retrieved. The figure below shows the summary of features from benign and malicious samples.

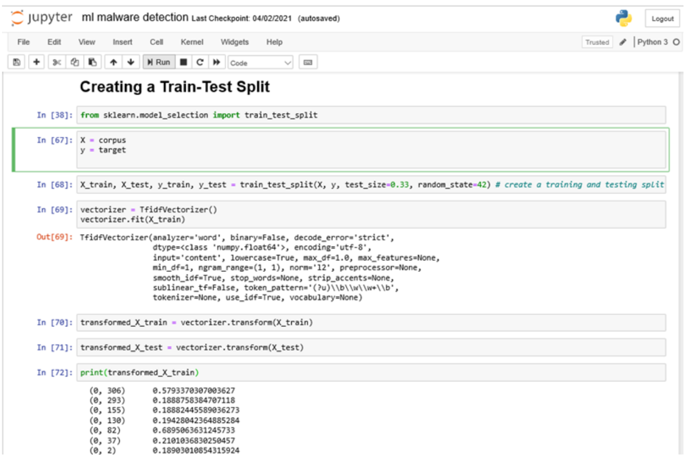
Step 3: Building the ML Security-Based Model

Generally, a security model can be a binary classification model or multi-class classification model.

The goal is to create a classifier that will read in the list of DLLs and then use it to predict whether a sample is benign or malicious. While TF records the number of occurrences of a word, IDF attaches significance to words by given small weights to common words and larger weights to important words.

a) Create a Train Test Split

We split our data into training and testing subsets. With that, we can fit our classifier to a training set and test it with the test set without the two overlappings. With sklearn, we assigned X as the corpus (features extracted) and y as the target (vector of 0’s and 1’s). The figure below shows the creation of the Train Test split of our samples.



Creating a Train-Test Split

```
In [38]: from sklearn.model_selection import train_test_split

In [67]: X = corpus
         y = target

In [68]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42) # create a training and testing split

In [69]: vectorizer = TfidfVectorizer()
         vectorizer.fit(X_train)

Out[69]: TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                         dtype=<class 'numpy.float64'>, encoding='utf-8',
                         input='content', lowercase=True, max_df=1.0, max_features=None,
                         min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=None,
                         smooth_idf=True, stop_words=None, strip_accents=None,
                         sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                         tokenizer=None, use_idf=True, vocabulary=None)

In [70]: transformed_X_train = vectorizer.transform(X_train)

In [71]: transformed_X_test = vectorizer.transform(X_test)

In [72]: print(transformed_X_train)

           (0, 306)      0.5793370307003627
           (0, 293)      0.1888758384707118
           (0, 155)      0.18882445589036273
           (0, 130)      0.19428042364885284
           (0, 82)       0.6895063631245733
           (0, 37)       0.2101036830250457
           (0, 2)        0.18903010854315924
```

Fig.4-12 Creating a Train Test Split

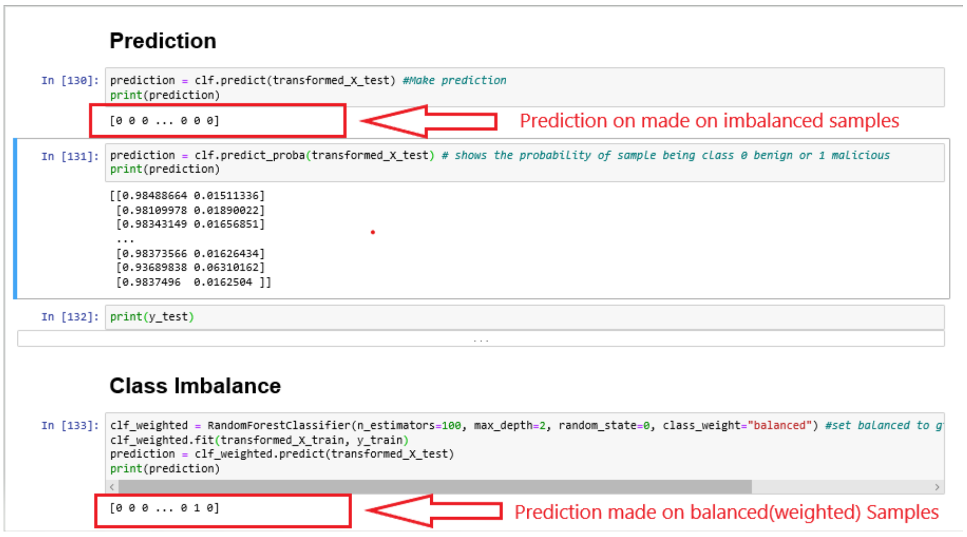
b) Train the Classifier

For the purpose of our work, we used the Random Forest classifier to train our classifier although we tried different kinds of classifiers (Gradient Boosting, etc.) to determine which performed best.

Step 4: Tackle Class Imbalances

In instances where there is an imbalance in the class (Available data sets), this can lead to error and reduction in the accuracy of the model.

Because our Data set is highly imbalanced as the ratio of Benign samples to Malicious samples is very high about 30:1, the model has made a prediction of all samples to be benign as seen below. The figure below shows a prediction made on our trained classifier.



Prediction

```
In [130]: prediction = clf.predict(transformed_X_test) #Make prediction
          print(prediction)

          [0 0 0 ... 0 0 0]
```

```
In [131]: prediction = clf.predict_proba(transformed_X_test) # shows the probability of sample being class 0 benign or 1 malicious
          print(prediction)

          [[0.98488664 0.01511336]
           [0.98109978 0.01890022]
           [0.98343149 0.01656851]
           ...
           [0.98373566 0.01626434]
           [0.93689838 0.06310162]
           [0.9837496  0.0162504 ]]

In [132]: print(y_test)
```

Class Imbalance

```
In [133]: clf_weighted = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0, class_weight="balanced") #set balanced to g
          clf_weighted.fit(transformed_X_train, y_train)
          prediction = clf_weighted.predict(transformed_X_test)
          print(prediction)

          [0 0 0 ... 0 1 0]
```

Fig.4-13 Predictions made on Imbalanced and balanced Samples

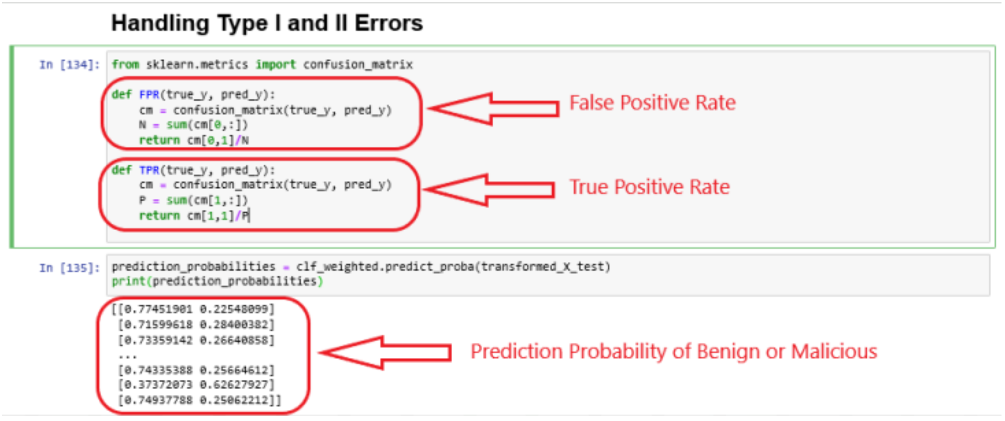
Since it was difficult to get more data to balance the ratio of our samples, we used the technique of adding more weights to the samples that are limited in number to increase our accuracy.

Step 5: Model Validation

a. Handling Type I and Type II Errors

It is important we took measures against the constraints of Type I and Type 2 errors thus false alarms (False Positive Rate (FPR)). Example when our classifier(model) wrongly classifies benign samples as malicious and malicious samples as benign. However, we do not want more than 10% of benign samples to be flagged as malicious thus we want our FPR < 0.1 or less. We also want to maximize our True Positives Rate (TPR) to get the largest coverage of malware.

To accomplish the task of reducing the FPR, we first get our predictions using the probability function of sklearn to determine the probability of a sample being either benign or malicious, then we set the threshold for the matrix which is used for the determination of benign and malicious samples i.e., if the threshold is set to 4, any value below the threshold is getting closer to 0 then classified as benign while any value above the threshold will be running to one then classified as malicious.



Handling Type I and II Errors

```
In [134]: from sklearn.metrics import confusion_matrix

          def FPR(true_y, pred_y):
              cm = confusion_matrix(true_y, pred_y)
              N = sum(cm[0,:])
              return cm[0,1]/N

          def TPR(true_y, pred_y):
              cm = confusion_matrix(true_y, pred_y)
              P = sum(cm[1,:])
              return cm[1,1]/P
```

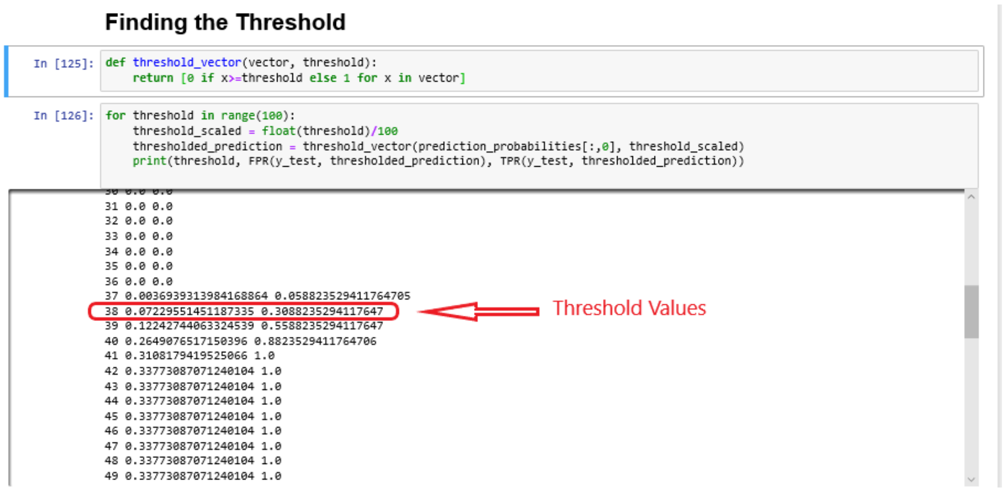
```
In [135]: prediction_probabilities = clf_weighted.predict_proba(transformed_X_test)
          print(prediction_probabilities)

          [[0.77451901 0.22548099]
           [0.71599618 0.28400382]
           [0.73359142 0.26640858]
           ...
           [0.74335388 0.25664612]
           [0.37372073 0.62627927]
           [0.74937788 0.25062212]]
```

Fig.4-14 Handling Type I and Type II Errors

In setting our threshold values we used the test set; it is important to use a validation set during iteration for the threshold value thus tune our parameters on a validation set and leave the test set for usage at the end.

In figure 4-15 below the threshold value, the True Positive Rate is maximized while the False Positive Rate is minimized below 0.1.



Finding the Threshold

```
In [125]: def threshold_vector(vector, threshold):
              return [0 if x>=threshold else 1 for x in vector]

In [126]: for threshold in range(100):
              threshold_scaled = float(threshold)/100
              thresholded_prediction = threshold_vector(prediction_probabilities[:,0], threshold_scaled)
              print(threshold, FPR(y_test, thresholded_prediction), TPR(y_test, thresholded_prediction))
```

```
31 0.0 0.0
32 0.0 0.0
33 0.0 0.0
34 0.0 0.0
35 0.0 0.0
36 0.0 0.0
37 0.0036939313984168864 0.058823529411764705
38 0.07229551451187335 0.3088235294117647
39 0.12242744063324539 0.5588235294117647
40 0.2649076517150396 0.8823529411764706
41 0.3108179419525066 1.0
42 0.33773087071240104 1.0
43 0.33773087071240104 1.0
44 0.33773087071240104 1.0
45 0.33773087071240104 1.0
46 0.33773087071240104 1.0
47 0.33773087071240104 1.0
48 0.33773087071240104 1.0
49 0.33773087071240104 1.0
```

Fig.4-15 Finding the Threshold Value

In instances where there is an imbalance in the class (Available data sets), this can lead to error and reduction in the accuracy of the model.

b. Performance Metrics

In simple terms, they define whether a model performs good or bad. This should be directly aligned with the higher-level goals of the business at hand as well as closely related to the kind of problem being solved. Example:

The regression problem uses mean squared error (MSE), R-square, and RMSE while Classification problems use precision, accuracy, sensitivity, MCC, and recall.

Insufficient data to learn from leads to poor accuracy of the model's learning algorithm. This can be improved by increasing the amount of training data examples and also the number of passes on existing training data.

4.4 N-grams to Classify Malware Implementation with ML

We used N-grams for the classification of malware with no domain knowledge on the samples. In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words, or base pairs according to the application. The n-grams typically are collected from a text or speech corpus. When the items are words, n-grams may also be called shingles [113]. They are used to capture a huge amount of local statistical information. We use the local statistics of N-gram to classify our malware. The following steps were taken:

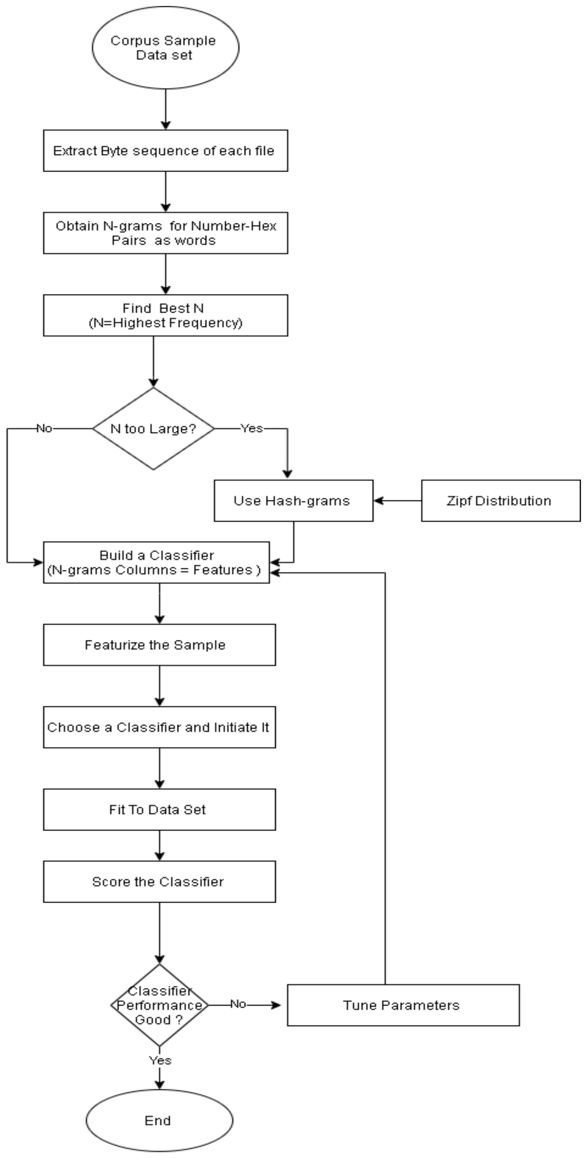
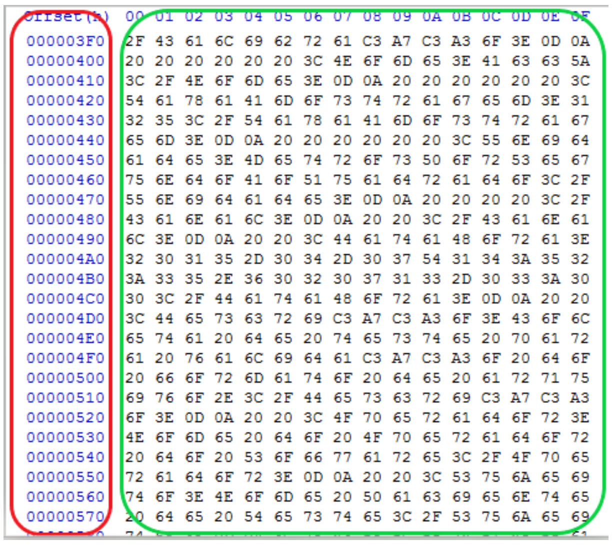

Corpus Sample Data set
Extract Byte sequence of each file
Obtain N-grams for Number-Hex Pairs as words
Find Best N (N=Highest Frequency)
N too Large?
No
Yes
Use Hash-grams
Zipf Distribution
Build a Classifier (N-grams Columns = Features)
Featurize the Sample
Choose a Classifier and Initiate It
Fit To Data Set
Score the Classifier
Classifier Performance Good ?
No
Tune Parameters
Yes
End

Fig.4-16 Implementation of N-gram for Malware Detection Architecture

Step 1: Examine the byte sequence of each sample

Using python, jupyter notebook, and TensorFlow we read in each file and extracted the byte sequence from each sample file.

From the figure below, the area marked red is the Address while the portion of the byte sequence of the file marked green is the sequence of Hex number pairs.



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000003F0  2F 43 61 6C 69 62 72 61 C3 A7 C3 A3 6F 3E 0D 0A
00000400  20 20 20 20 20 20 3C 4E 6F 6D 65 3E 41 63 63 5A
00000410  3C 2F 4E 6F 6D 65 3E 0D 0A 20 20 20 20 20 20 3C
00000420  54 61 78 61 41 6D 6F 73 74 72 61 67 65 6D 3E 31
00000430  32 35 3C 2F 54 61 78 61 41 6D 6F 73 74 72 61 67
00000440  65 6D 3E 0D 0A 20 20 20 20 20 20 3C 55 6E 69 64
00000450  61 64 65 3E 4D 65 74 72 6F 73 50 6F 72 53 65 67
00000460  75 6E 64 6F 41 6F 51 75 61 64 72 61 64 6F 3C 2F
00000470  55 6E 69 64 61 64 65 3E 0D 0A 20 20 20 20 3C 2F
00000480  43 61 6E 61 6C 3E 0D 0A 20 20 3C 2F 43 61 6E 61
00000490  6C 3E 0D 0A 20 20 3C 44 61 74 61 48 6F 72 61 3E
000004A0  32 30 31 35 2D 30 34 2D 30 37 54 31 34 3A 35 32
000004B0  3A 33 35 2E 36 30 32 30 37 31 33 2D 30 33 3A 30
000004C0  30 3C 2F 44 61 74 61 48 6F 72 61 3E 0D 0A 20 20
000004D0  3C 44 65 73 63 72 69 C3 A7 C3 A3 6F 3E 43 6F 6C
000004E0  65 74 61 20 64 65 20 74 65 73 74 65 20 70 61 72
000004F0  61 20 76 61 6C 69 64 61 C3 A7 C3 A3 6F 20 64 6F
00000500  20 66 6F 72 6D 61 74 6F 20 64 65 20 61 72 71 75
00000510  69 76 6F 2E 3C 2F 44 65 73 63 72 69 C3 A7 C3 A3
00000520  6F 3E 0D 0A 20 20 3C 4F 70 65 72 61 64 6F 72 3E
00000530  4E 6F 6D 65 20 64 6F 20 4F 70 65 72 61 64 6F 72
00000540  20 64 6F 20 53 6F 66 77 61 72 65 3C 2F 4F 70 65
00000550  72 61 64 6F 72 3E 0D 0A 20 20 3C 53 75 6A 65 69
00000560  74 6F 3E 4E 6F 6D 65 20 50 61 63 69 65 6E 74 65
00000570  20 64 65 20 54 65 73 74 65 3C 2F 53 75 6A 65 69
```

Fig.4-17 Byte Sequence of a file

Step 2: The hex-number pairs are used as words

We used python library NLTK used for natural language processing to obtain the actual N-gram.

Step 3: Find the best N

The best N depends on the application. When N is small much information of the local statistics is not capture and when N is very large it will be overfitting thus capturing too much information. Therefore, cannot generalize well.

The best or most important N-grams are the most frequent ones. We, therefore, selected the top k (i.e., 1000 N-grams).

Step 4: Determine the useful N-grams and discard the none useful ones

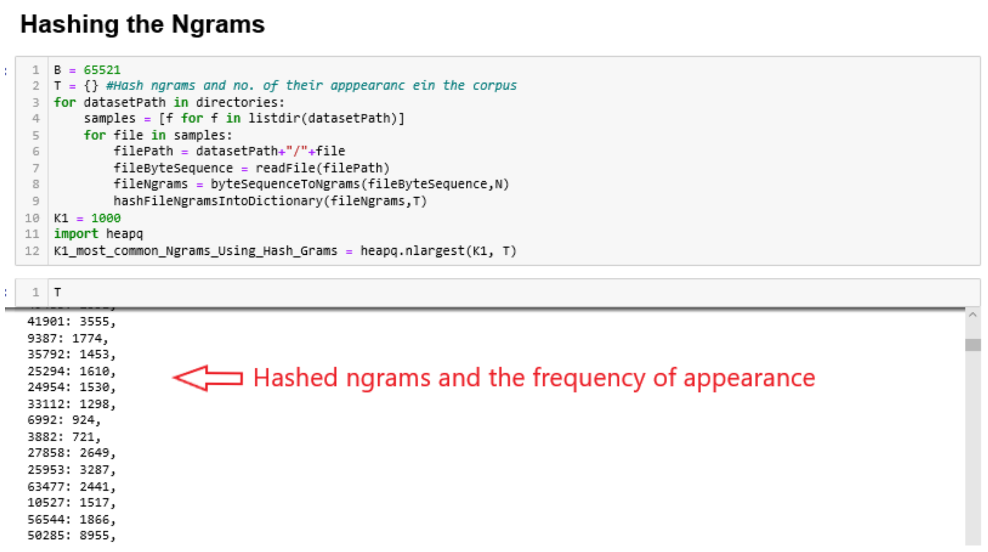
As the number of N-Grams grows exponentially, it causes a problem of storage, computational time and can decrease the accuracy of the classifier.

Hash-grams:

We used this technique to overcome the problem of running out of memory, by hashing all the N-grams as they commutatively pile up in the dictionary.

Zipf Distribution [114].

Our most frequent N-grams which we are only interested in follow the Zipf distribution indicating the frequent ones are few while the infrequent ones are very many. The figure below shows the extraction and count of hashed n-grams and the number of their appearances.



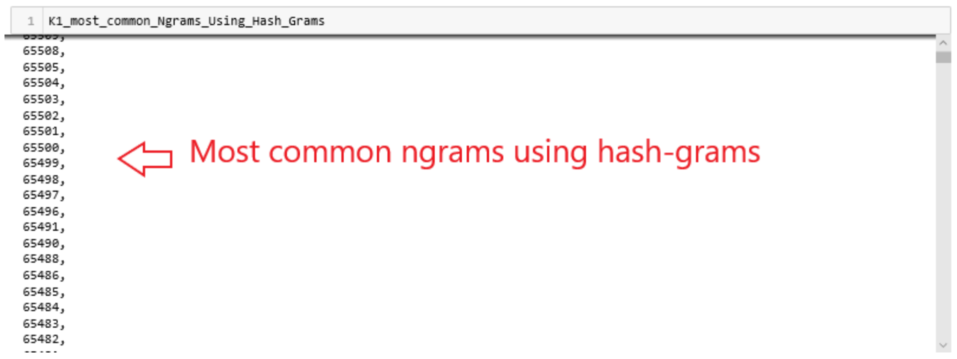
Hashing the Ngrams

```
B = 65521
T = {} #Hash ngrams and no. of their apppearanc ein the corpus
for datasetPath in directories:
    samples = [f for f in listdir(datasetPath)]
    for file in samples:
        filePath = datasetPath+"/"+file
        fileByteSequence = readFile(filePath)
        fileNgrams = byteSequenceToNgrams(fileByteSequence,N)
        hashFileNgramsIntoDictionary(fileNgrams,T)
K1 = 1000
import heapq
K1_most_common_Ngrams_Using_Hash_Grams = heapq.nlargest(K1, T)
```

```
T
```

```
41901: 3555,
9387: 1774,
35792: 1453,
25294: 1610,
24954: 1530,
33112: 1298,
6992: 924,
3882: 721,
27858: 2649,
25953: 3287,
63477: 2441,
10527: 1517,
56544: 1866,
50285: 8955,
```

Fig.4-18 Hashed ngrams and their frequencies of appearance.



```
K1_most_common_Ngrams_Using_Hash_Grams
```

```
65508,
65505,
65504,
65503,
65502,
65501,
65500,
65499,
65498,
65497,
65496,
65491,
65490,
65488,
65486,
65485,
65484,
65483,
65482,
```

Fig.4-19 Most common ngrams using hash-grams.

Step 5: Build a classifier using N-grams columns as features and predict malware

a. Build the list of Hash-grams and their frequencies in the corpus from the Hash-gram algorithm.

b. Extract the features- Take the top k: k=100 Hash-grams by frequency.

c. Futurize the samples:

1. Read in each sample and check if it contains each of the hashed N-gram. If it contains, it's assigned to 1 else 0.

2. Create a target array: Each label corresponds to a different malware family. Number of labels = 9 (This improves the performance score of the classifier)

3. Loop over all samples to collect; Feature vectors in the training into the Train_X set and Labels in the Train_Y set.

Step 7: Choose a classifier and instantiate it

We used DecisionTreeClassifier for the purpose of our work.

Step 8: Fit to Training Data

Fit the Data to the classifier.

Step 9: Score the classifier under the training set

Creating labels improve accuracy far better than using a baseline of random assigning of labels.

Step 11: Set out the Testing Data set:

Futurize the Testing Data and giving it the labels.

Step 12: Fit Classifier to the Testing Data Set:

Futurize the Testing Data and giving it the labels.

Step 13: Score the classifier under the testing set:

Overfitting may occur If the classifier performed well on the training set but poorly on the testing set. To correct the overfitting;

A different classifier (i.e., Random Forest, Gradient Boosting) may be tried or adding regularization or trying different N-grams.

4.5 Deep Learning to Classify Malware Implementation

Using Keras and TensorFlow we explore the implementation of an end-to-end technique deep learning classifier on our raw samples. The diagram below gives the general structure of the convolution Neural Network used in our experiment for malware detection using deep learning.

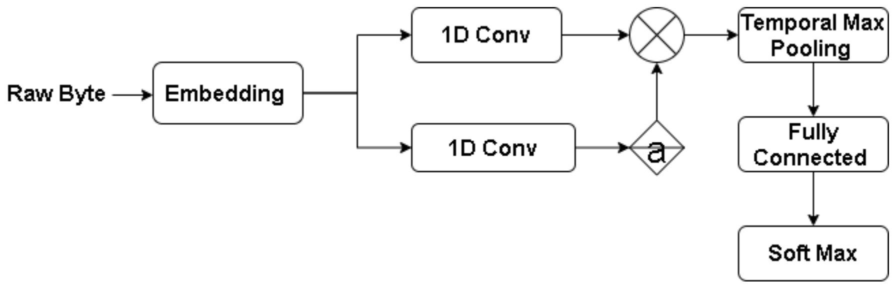


Fig.4-20 Convolution Neural Network Scheme for Malware Detection

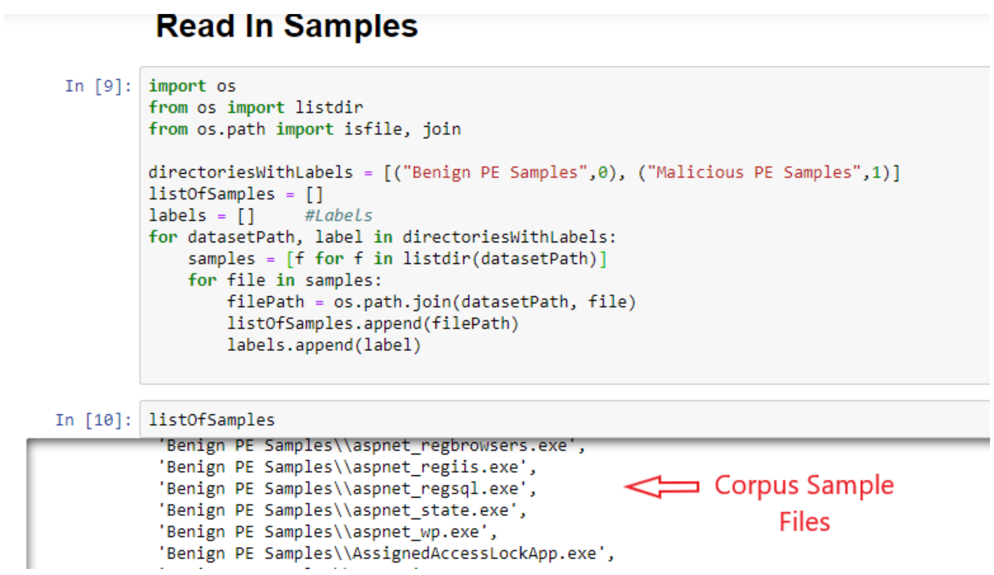
Our task is to create a Neural Network to classify our samples (byte sequence format) to any of the labels (nine types of malware)

The following steps were followed:

Step 1: Compiling the DL Model

a. Read in pure byte sequence of the samples:

The byte sequence is interpreted as a series of the scan to a video or audio stream. Our Neural Network will be classified into one of the 9-types of malware.



Read In Samples

In [9]:
```
import os
from os import listdir
from os.path import isfile, join

directoriesWithLabels = [("Benign PE Samples",0), ("Malicious PE Samples",1)]
listOfSamples = []
labels = []      #Labels
for datasetPath, label in directoriesWithLabels:
    samples = [f for f in listdir(datasetPath)]
    for file in samples:
        filePath = os.path.join(datasetPath, file)
        listOfSamples.append(filePath)
        labels.append(label)
```

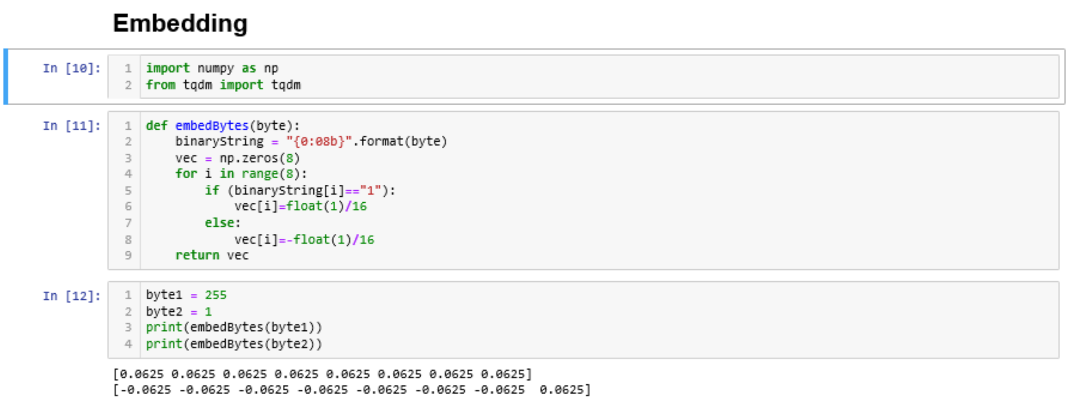
In [10]:
```
listOfSamples
```
```
 'Benign PE Samples\\aspnet_regbrowsers.exe',
 'Benign PE Samples\\aspnet_regiis.exe',
 'Benign PE Samples\\aspnet_regsql.exe',
 'Benign PE Samples\\aspnet_state.exe',
 'Benign PE Samples\\aspnet_wp.exe',
 'Benign PE Samples\\AssignedAccessLockApp.exe',
```

Fig.4-21 Benign and Malicious Executables Sample Files

b. Embedding - Convert the inputs into a numerical form (Vectorizing the Data);

By embedding each byte sequence into an 8-dimensional vector. There are various types of embedding, an example is Word2Vec, etc. For our work, we used a Simple Embedding mechanism. For each byte in binary form, we make a replacement of; 1=1/16 and 0=-1/16.

c. Using python and jupyter notebook we iterated through all the files of our Benign and Malicious samples in their byte format and performed the embedding.



Embedding

In [10]:
```
import numpy as np
from tqdm import tqdm
```

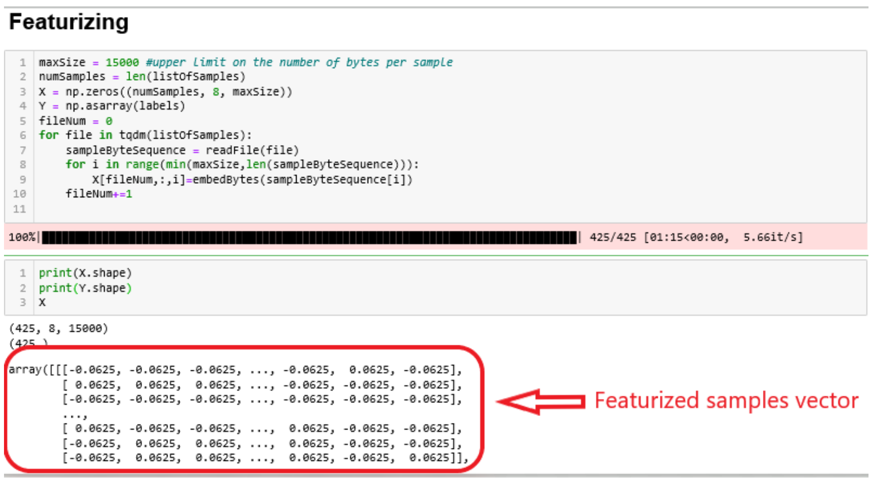
In [11]:
```
def embedBytes(byte):
    binaryString = "{0:08b}".format(byte)
    vec = np.zeros(8)
    for i in range(8):
        if (binaryString[i]=="1"):
            vec[i]=float(1)/16
        else:
            vec[i]=-float(1)/16
    return vec
```

In [12]:
```
byte1 = 255
byte2 = 1
print(embedBytes(byte1))
print(embedBytes(byte2))
```
```
[0.0625 0.0625 0.0625 0.0625 0.0625 0.0625 0.0625 0.0625]
[-0.0625 -0.0625 -0.0625 -0.0625 -0.0625 -0.0625 -0.0625  0.0625]
```

Fig.4-22 Embedding the Byte to Numerical form (Vectorizing)

Step 2: Read in the Samples

We Set an upper limit on the number of bytes that will be reading in per sample: This is to ensure a uniform size for all the samples and also to overcome the computational constraint.



Featurizing

```
maxSize = 15000 #upper Limit on the number of bytes per sample
numSamples = len(listOfSamples)
X = np.zeros((numSamples, 8, maxSize))
Y = np.asarray(labels)
fileNum = 0
for file in tqdm(listOfSamples):
    sampleByteSequence = readFile(file)
    for i in range(min(maxSize,len(sampleByteSequence))):
        X[fileNum,:,i]=embedBytes(sampleByteSequence[i])
    fileNum+=1

```

```
100%|████████████████████████████████████████████████████████████| 425/425 [01:15<00:00,  5.66it/s]
```

```
print(X.shape)
print(Y.shape)
X
```

```
(425, 8, 15000)
(425,)
array([[[-0.0625, -0.0625, -0.0625, ..., -0.0625,  0.0625, -0.0625],
        [ 0.0625,  0.0625,  0.0625, ..., -0.0625, -0.0625, -0.0625],
        [-0.0625, -0.0625, -0.0625, ..., -0.0625, -0.0625, -0.0625],
        ...,
        [ 0.0625, -0.0625, -0.0625, ...,  0.0625, -0.0625, -0.0625],
        [-0.0625,  0.0625,  0.0625, ...,  0.0625, -0.0625, -0.0625],
        [-0.0625,  0.0625,  0.0625, ...,  0.0625, -0.0625,  0.0625]],
```

Fig.4-23 Futurized samples into vector

Step 3: Create a Training Data set

a. Creating Train_X Set: each byte is embedded and then collected into the Train_X set.

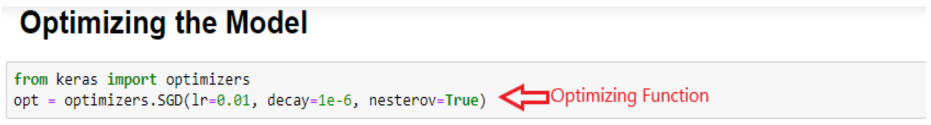
b. Create Train_Y set: This is assigned to the labels available in the data set.

Step 4: Convert labels to a categorical data

We used One hot technique for the conversion of the Target(labels) to a categorical data form from 0-8 for the Neural Network.

Step 5: Create the Neural Network with Keras.

a. Declare the optimizer – for example Stochastic Gradient Descent.



Optimizing the Model

```
from keras import optimizers
opt = optimizers.SGD(lr=0.01, decay=1e-6, nesterov=True)
```

Fig.4-24 Optimization of the Neural Network with Keras

b. Specify the shape of the input:

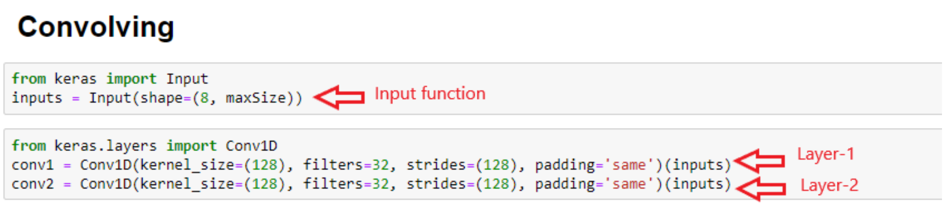
State the length(maxsize) of the vector. Use padding if the file is too short for maximum size.

c. Feed inputs to the convolutional functions with their parameters and using the Keras functional API i.e., conv1, conv2, etc. Creating the functions gives Keras give an advantage of flexibility for all sorts of combinations.

d. Convolving Inputs and Combing Outputs

Activation 1:

i. Use example Sigmoid or any other function to activate only one of the convolutional function example; conv2.



Convolving

```
from keras import Input
inputs = Input(shape=(8, maxSize))
```

```
from keras.layers import Conv1D
conv1 = Conv1D(kernel_size=(128), filters=32, strides=(128), padding='same')(inputs)
conv2 = Conv1D(kernel_size=(128), filters=32, strides=(128), padding='same')(inputs)
```

Fig.4-25 Convolving of the Neural Network with Keras

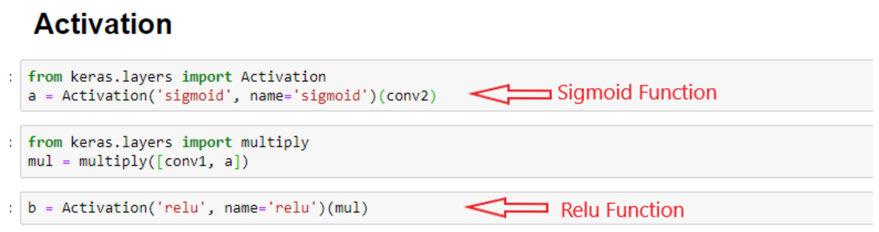
ii. Multiply of Function

Multiply both the activated and non-activated functions.

iii. Activate the results of the product from the multiplication.

e. Neural Network Activation 2:

i. Update the activation variable by Activating the results of the multiplication. Example use relu function or any other that is suitable.



Activation

```
from keras.layers import Activation
a = Activation('sigmoid', name='sigmoid')(conv2)
```

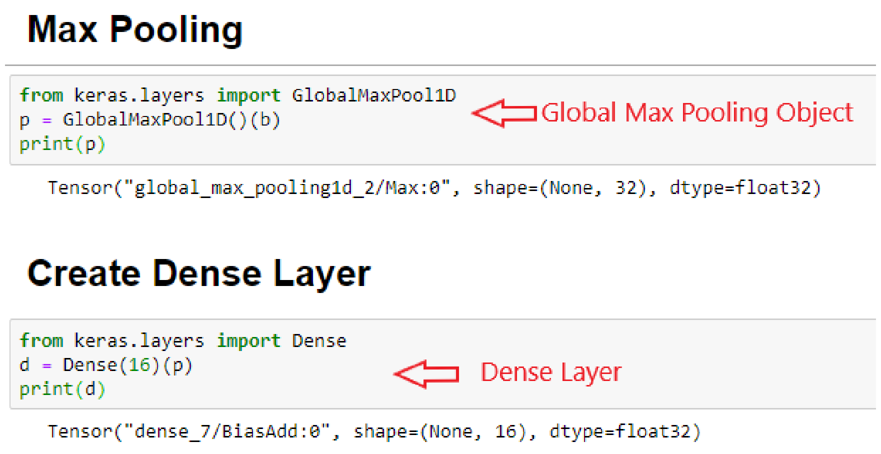
```
from keras.layers import multiply
mul = multiply([conv1, a])
```

```
b = Activation('relu', name='relu')(mul)
```

Fig.4-26 Activation of the Neural Network with Keras

f. Create a GlobalMaxPool on the new activation variable created.

g. Create the Dense layer with the GlobalMax pool variable.



Max Pooling

```
from keras.layers import GlobalMaxPool1D
p = GlobalMaxPool1D()(b)
print(p)
```

```
Tensor("global_max_pooling1d_2/Max:0", shape=(None, 32), dtype=float32)
```

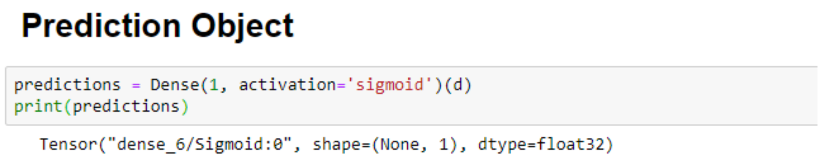
Create Dense Layer

```
from keras.layers import Dense
d = Dense(16)(p)
print(d)
```

```
Tensor("dense_7/BiasAdd:0", shape=(None, 16), dtype=float32)
```

Fig.4-27 Max Pooling and Dense Layer Created with Keras

h. Create a Prediction object: use the dense layer to finally make the prediction.



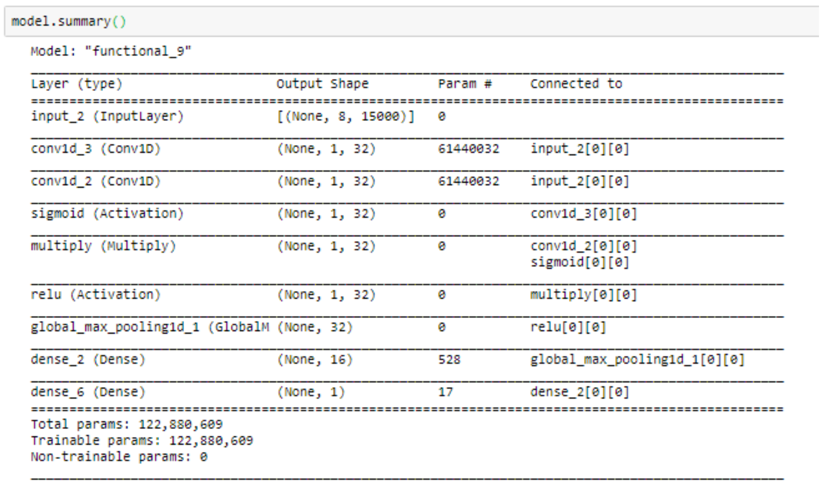
Prediction Object

```
predictions = Dense(1, activation='sigmoid')(d)
print(predictions)
```

```
Tensor("dense_6/Sigmoid:0", shape=(None, 1), dtype=float32)
```

Fig.4-28 Prediction Object Created

i. Compile the model with the prediction made; Check the summary



```
model.summary()
```

```
Model: "functional_9"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 8, 15000)]   0
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 1, 32)        61440032    input_2[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 1, 32)        61440032    input_2[0][0]
__________________________________________________________________________________________________
sigmoid (Activation)            (None, 1, 32)        0           conv1d_3[0][0]
__________________________________________________________________________________________________
multiply (Multiply)             (None, 1, 32)        0           conv1d_2[0][0]
                                                                 sigmoid[0][0]
__________________________________________________________________________________________________
relu (Activation)               (None, 1, 32)        0           multiply[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 32)           0           relu[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 16)           528         global_max_pooling1d_1[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 1)            17          dense_2[0][0]
==================================================================================================
Total params: 122,880,609
Trainable params: 122,880,609
Non-trainable params: 0
__________________________________________________________________________________________________
```

Fig.4-29 Model Summary

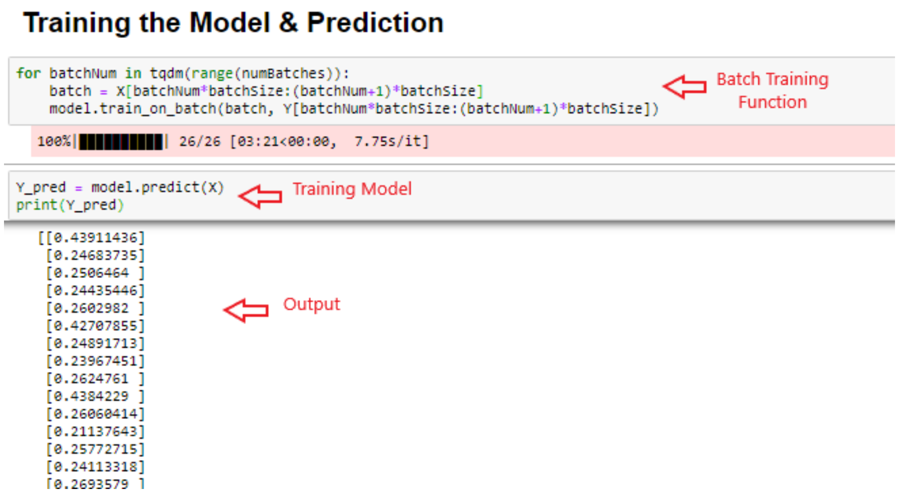
Step 6: Train the Neural Network Classifier

a. Split samples into batches; Train the model: one batch at a time

b. Predict on the training set: to check the fit; Prediction gives an array of unnormalized probabilities, the highest number per vector corresponds to the most likely label.

c. Iterate through each row(vector) pick the highest number as the prediction.

d. Compare with the actual labels to get the accuracy.



Training the Model & Prediction

```
for batchNum in tqdm(range(numBatches)):
    batch = X[batchNum*batchSize:(batchNum+1)*batchSize]
    model.train_on_batch(batch, Y[batchNum*batchSize:(batchNum+1)*batchSize])
```

```
100%|██████████| 26/26 [03:21<00:00,  7.75s/it]
```

```
Y_pred = model.predict(X)
print(Y_pred)
```

```
[[0.43911436]
 [0.24683735]
 [0.2506464 ]
 [0.24435446]
 [0.2602982 ]
 [0.42707855]
 [0.24891713]
 [0.23967451]
 [0.2624761 ]
 [0.4384229 ]
 [0.26060414]
 [0.21137643]
 [0.25772715]
 [0.24113318]
 [0.2693579 ]
```

Fig.4-30 Training and Predict with Model

Step 7: Testing the Neural Network Classifier

Repeat the same procedure of the training but with the Test set to make a prediction.

4.6 In a Production Environment

To increase the accuracy for production, the following measures may be taken;

i. Tune parameter

ii. Increase data science

iii. Check class bounds Using Keras and TensorFlow we explore the implementation of end-to-end technique deep learning classifier on our raw samples

4.7 Summary

In this section, we completed 3 experiments for the application of machine learning in malware detection leveraging on Cybersecurity Data Science. The three mechanisms were Using the PE file DLL information, N-grams, and Deep Learning.

5 EXPERIMENT DISCUSSION AND RESULTS ANALYSIS

In this section, we discuss and analyze our experiment and its results.

5.1 Environment Setup

We set up our lab's environments, obtain samples, perform Feature engineering, build classifiers on top of the features, Access the performance of the classifiers. With our raw sample, we explored sophisticated mechanisms such as PE Files DLLs, N-grams and deep neural network which does not rely on any feature.

5.2 Environment Configuration

Tab.5-1 Experimental environment configuration

Experimental environment	Configuration
Host machine Operation System	Windows 10 Home 64-bit
CPU	Inter(R) Core(TM) i7-8550U CPU @1.80GHz (* CPUs)
Memory	16G
Hard Disk	1T
Programming Language	Python 3.5.6
Deep Learning Framework	Keras
Framework Background	TensorFlow 2.5

5.2.1 Tools and Installations

a) Virtualization software: Oracle VirtualBox 6.1

b) Virtual System configuration: MSEDGE

c) Host Operating System: Windows 10 Home

d) Windows Virtual Image: MSEdge on Win10(x64) Stable 1809 VM Image

e) Windows Tool String

f) Python programming language

g) Jupyter Notebook

h) Anaconda

5.2.2 Precautions

a) Malware files must be analyzed in a controlled environment to avoid the risk of infections

b) Disconnection from Network: As earlier indicated, working on malware needs to be done in a seriously controlled environment. It is therefore important to disconnect the VM from the network or the host machine to prevent the risk of unintentionally infecting the whole network with malware.

c) Disabling Windows Defender: It is important to disable Windows defender on VM while downloading the malware, else windows will flag the malware before the downloading completes.

d) Shared Folder: A shared folder needs to be created so files can be transferred between the host machine and the guest virtual machine.

e) To avoid the risk of infection, it’s very important to work on malware in a controlled environment. VirtualBox can be downloaded on www.virtualbox.org   and installed.

5.3 Experiment Comparison Analysis

5.3.1 Data Set

The Data set for our experiment was from open-source public data; Github and Kaggle. We also generated our Benign Data set sample locally from our system file. However, in the process of generating the sample locally from a local source machine, Extra care needs to be taken in order not to add contaminate the samples into the benign samples.

Tab.5-2 Data Set and Its Features

Category	Samples Files	Corpus Sections
Benign Corpus	336	5748
Malicious Corpus	44	198
Total	380	5946

5.4 Summary

In this chapter, we made an exclusive discussion and analysis of all the three techniques of applying machine learning to malware detection

6 CONCLUSIONS

Cyber-attacks are always and constantly evolving, no one can know and can determine the form it will take in the Future. Data Science and Machine Learning are playing a significant role in today’s world of the fight against malware and cyber-attacks through prediction of the future threat using historical data. Also, a lot of works have been than in the field of application of Machine Learning and Deep learning in malware detection and cybersecurity. Although it’s tempting to consider the Data-Driven approach to the fight against malware as the panacea for all problems, it is important to cautious and takes cognizance of the pitfalls in applying this approach to a cybersecurity problem.

6.1 Work Summary

In this paper we demonstrate the application and implementation of Machine learning techniques for the detection and classification of malware, using both supervised and unsupervised machine learning techniques for anomaly detection in cybersecurity. We emphasize the description of machine learning workflow and methods of their usage in the context of cybersecurity, machine learning and data science.

Our work is mainly an exploratory implementation of machine learning techniques for malware detection using three mechanisms; The P.E files DLL, N-grams and Deep Learning on executable sample. In this paper we demonstrate the application and implementation of Machine Learning techniques for the detection and classification of malware, using both supervised and unsupervised machine learning techniques for the classification of malware leveraging cybersecurity data science. We emphasize the description of machine learning workflow and methods of their usage in the context of cybersecurity data science.

The main contribution of our work is summarized as follows;

1) Demonstrated the application of three(3) different machine learning techniques to classify and predict malware.

2) Discuss the Machine Learning technique for Malware Classification

3) Examine the previous works done on the research on malware detections in the direction of a data-driven approach in the domain of cybersecurity.

4) Present a comprehensive guide of the framework and workflow of machine learning in the context of cybersecurity data science for malware detection.

6.2 Work Prospects and Future Expectation

Although machine learning has been a great tool in the fight against malware attacks, our work opens several research issues and challenges in the area of cybersecurity data science to extract insight from relevant data towards data-driven intelligent decision-making for cybersecurity solutions. We summarize these challenges ranging from data collection to decision making which includes:

a. Threats of Adversarial attacks: recent works in this area have indicated that attackers have been able to bypass the machine learning malware detection models, through the construction of specific samples. Wu et al in their presentation indicated that, according to the degree of adversaries' knowledge about the model, adversarial attacks can be classified into several groups, such as gradient and score-based attacks. In their paper, they propose a more general framework based on deep reinforcement learning (DRL), which effectively generates adversarial traffic flows to deceive the detection model by automatically adding perturbations to samples. Throughout the process, the target detector will be regarded as a black box and closer to realistic attack circumstances. A reinforcement learning agent is equipped for updating the adversarial samples by combining the feedback from the target model (i.e., benign or malicious) and the sequence of actions, which can change the temporal and spatial features of the traffic flows while maintaining the original functionality and executability. Their experiment results showed that the evasion rates of adversarial botnet flows are significantly improved. Furthermore, with the perspective of defense, their research can help the detection model spot its defect and thus enhance the robustness [115].

b. Feature engineering in cybersecurity: The efficiency of an ML model is highly dependent on its feature derived from the Data set fed to the model. As the efficiency and effectiveness of a machine learning-based security model have always been a major challenge due to the high volume of network data with a large number of traffic features.

c. Handling quality problems in cybersecurity datasets: The cyber datasets might be noisy, incomplete, insignificant, imbalanced, or may contain inconsistent instances related to a particular security incident.

Acknowledgments

It is my greatest honor to further my studies in master’s degree at the School of Computer Science and Engineering at the Xi’an University of Technology.

My most important gratification goes to my Creator almighty God, for giving me the strength of good health, knowledge, and wisdom to undertake this program and his continued mercies and guidance of my life.

My sincere gratitude goes to my supervisor, Professor Sun Qindong, for his dedication patience, and commitment to not only tutoring and guidance but also supporting, and mentoring me for my entire period of studies in the University. I cannot thank Professor Sun enough for all his sacrifices to make time and be available to me at any time I needed his help and assistance, despite his busy schedules. His encouragement, inspiration, and patient mentoring are a permanent mark in my life which I can never forget.

I also want to give thanks to Professor Zhao Minghua for giving me the opportunity to work on two research projects whiles during my period of study. Professor Zhao’s patient guidance, meticulous approach to issues and problems were a source of great motivation during my years of study at the University.

Most importantly I could not have come this far without the support of my family, especially my beloved Mother for giving me all the inspiration, emotional and spiritual motivation, and all other support I ever needed.

I am also grateful to Wang Qin and the other Chinese student of Computer Science and more importantly my laboratory mates and friends for constant support and advice during my academic life and studies experience as a foreign student of the Xi’an University of Technology.

Furthermore, I wish to extend appreciation to the office for International Students of the University for their logistical support during the period of studies.

To conclude I thank all the teachers who have spent effort and time in reviewing my thesis, their corrections and recommendation are highly appreciated and welcome and will be implemented accordingly.

Reference

[1] Gustafsson B. The Beginning of the Computer Era. In Springer, Cham; 2018 [J] p. 89–132. Available from: https://link.springer.com/chapter/10.1007/978-3-319-69847-2_4

[2] Sun Q, Cao H, Qi W, Zhang J. Improving the security and quality of real-time multimedia transmission in cyber-physical-social systems.[J] Int J Distrib Sens Networks [Internet]. 2018 Nov 3;14(11):155014771881069. Available from: http://journals.sagepub.com/doi/10.1177/1550147718810694

[3] Robert Moir. Defining Malware: FAQ [N]. 2009. Available from: https://docs.microsoft.com/en-us/previous-versions/tn-archive/dd632948(v=technet.10)?redirectedfrom=MSDN

[4] Christodorescu M, Jha S, Seshia SA, Song D, Bryant RE. Semantics-aware malware detection. In: 2005 IEEE Symposium on Security and Privacy (S&P'05) [C]. IEEE; 2005 p. 32–46. Available from: https://ieeexplore.ieee.org/document/1425057/

[5] Chadd K. The History of Cybersecurity | Avast [N]. Avast. 2020. Available from: https://blog.avast.com/history-of-cybersecurity-avast#the-1970s

[6] Economics C. Annual Worldwide Economic Damages from Malware Exceed {$}13 Billion [J]. 2007. Available from: https://www.computereconomics.com/article.cfm?id=1225

[7] AV-Test. Malware Statistics & Trends Report | AV-TEST [R]. 2020 [cited 2021 Feb 16]. Available from: https://www.av-test.org/en/statistics/malware/

[8] Ye Y, Wang D, Li T, Ye D. IMDS. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07 [J]. New York, New York, USA: ACM Press; 2007. p. 1043. Available from: http://portal.acm.org/citation.cfm?doid=1281192.1281308

[9] Gibert D, Mateu C, Planes J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. J Netw Comput Appl [J]. 2020 Mar 1;153:102526. Available from: https://linkinghub.elsevier.com/retrieve/pii/S1084804519303868

[10] Liu X, Lin Y, Li H, Zhang J. A novel method for malware detection on ML-based visualization technique. Compute Secure [J]. 2020 Feb 1;89:101682. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0167404818314627

[11] Gandotra E, Bansal D, Sofat S. Malware Analysis and Classification: A Survey. J Inf Secur [J]. 2014 Feb 20;05(02):56–64. Available from: http://www.scirp.org/journal/jishttp://dx.doi.org/10.4236/jis.2014.52006http://dx.doi.org/10

.4236/jis.2014.52006http://creativecommons.org/licenses/by/4.0/

[12] O'Kane P, Sezer S, McLaughlin K. Obfuscation: The Hidden Malware. IEEE Secur Priv [J]. 2011 Sep;9(5):41–7. Available from: https://ieeexplore.ieee.org/document/5975134/

[13] Bayer U, Kirda E, Kruegel C. Improving the efficiency of dynamic malware analysis. In: Proceedings of the ACM Symposium on Applied Computing [J]. New York, New York, USA: ACM Press; 2010. p. 1871–8. Available from: http://portal.acm.org/citation.cfm?doid=1774088.1774484

[14] Sharma, A., K. Sahay, S., 2014. Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey. International Journal of Computer Applications[J] 90, 7–11.. doi:10.5120/15544-4098.

[15] Peter Tippett. The Fourth Generation of Malware -- CIO Update [N]. 2006. Available from: https://cioupdate.com/the-fourth-generation-of-malware/

[16] Swain B. What are malware, viruses, Spyware, and cookies, and what differentiates them ? [N]. Symantec Connect Community. 2009. Available from: https://www.websecurity.digicert.com/security-topics/what-are-malware-viruses-spyware-and-cookies-and-what-differentiates-them

[17] BBC. BBC Bitesize - What are viruses and malware? [N]. 2018. Available from: https://www.bbc.co.uk/bitesize/topics/zd92fg8/articles/zcmbgk7

[18] Nadeau M. What is cryptojacking? How to prevent, detect, and recover from it,. csoonline [N]. 2018;https://www.csoonline.com. Available from: https://www.csoonline.com/article/3253572/what-is-cryptojacking-how-to-prevent-detect-and-recover-from-it.html

[19] Top cryptomining malware worldwide 2020 | Statista [R]. Available from: https://www.statista.com/statistics/325252/cryptomining-malware-global/

[20] McAfee. What Is the Difference Between Malware and a Virus? [N]. McAfee.. Available from: https://www.mcafee.com/enterprise/en-us/security-awareness/ransomware/malware-vs-viruses.html

[21] Remote Access Trojan (RAT) | Kaspersky IT Encyclopedia [EB]. Available from: https://encyclopedia.kaspersky.com/glossary/remote-access-trojan-rat/

[22] Rezaeirad M, Farinholt B, Dharmdasani H, Pearce P, Levchenko K, McCoy D. Schrödinger's RAT: Profiling the stakeholders in the remote access Trojan ecosystem. In: Proceedings of the 27th USENIX Security Symposium [C]. 2018. p. 1043–60. Available from: https://www.usenix.org/conference/usenixsecurity18/presentation/rezaeirad

[23] Pu Y, Chen X, Cui X, Shi J, Guo L, Qi C. Data stolen trojan detection based on network behaviors.[J] In: Procedia Computer Science. Elsevier B.V.; 2013. p. 828–35.

[24] Computer Trojan Types | How Computer Trojan Viruses Work? [EB/OL].. Available from: https://enterprise.comodo.com/blog/computer-trojan-viruses-and-how-they-work/

[25] Trojan Horse Software Attack | OWASP Foundation [EB/OL].Available from: https://owasp.org/www-community/attacks/Trojan_Horse

[26] What Is a Sniffer? | How to Protect Yourself | AVG [EB/OL]. Available from: https://www.avg.com/en/signal/what-is-sniffer

[27] What is a keylogger and how do I help protect myself? | NortonLifeLock [EB/OL]. Available from: https://us.norton.com/internetsecurity-malware-what-is-a-keylogger.html

[28] What is a Botnet? - Palo Alto Networks [EB/OL]. paloaltonetworks.com. Available from: https://www.paloaltonetworks.com/cyberpedia/what-is-botnet

[29] Yener B, Gal T. Cybersecurity in the Era of Data Science: Examining New Adversarial Models.[J] IEEE Secur Priv. 2019 Nov 1;17(6):46–53.

[30] What is Workload? - Definition from Techopedia [EB/OL]. Techopedia.com. 2018. Available from: https://www.techopedia.com/definition/4120/spamware

[31] What Does A Malware Virus Do? | Different Types Of Malware [EB/OL]. Available from: https://enterprise.comodo.com/what-does-a-malware-virus-do.php

[32] El Naqa I, Murphy MJ. What Is Machine Learning? In: Machine Learning in Radiation Oncology [J]. Cham: Springer International Publishing; 2015. p. 3–11. Available from: https://www.ibm.com/cloud/learn/machine-learning

[33] Gavrilut D, Cimpoesu M, Anton D, Ciortuz L. Malware detection using machine learning. In: 2009 International Multiconference on Computer Science and Information Technology [J]. IEEE; 2009. p. 735–41. Available from: https://ieeexplore.ieee.org/document/5352759/

[34] Sharma S, Rama Krishna C, Sahay SK, Scholar M, Krishna CR, Sahay SK. Detection of Advanced Malware by Machine Learning Techniques. Adv Intell Syst Comput [J]. 2019 Mar 7;742:333–42. Available from: https://link.springer.com/chapter/10.1007/978-981-13-0589-4_31

[35] Ye Y, Li T, Chen Y, Jiang Q. Automatic malware categorization using cluster ensemble. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10 [J]. New York, New York, USA: ACM Press; 2010. p. 95. Available from: http://dl.acm.org/citation.cfm?doid=1835804.1835820

[36] Hansen J V, Lowry PB, Meservy RD, McDonald DM. Genetic programming for prevention of cyberterrorism through dynamic and evolving intrusion detection. Decis Support Syst [J]. 2007;43(4):1362–74. Available from: www.elsevier.com/locate/dss

[37] Yin C, Zhu Y, Fei J, He X. A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. IEEE Access [J]. 2017;5:21954–61. Available from:

https://ieeexplore.ieee.org/abstract/document/8066291/

[38] Almiani M, AbuGhazleh A, Al-Rahayfeh A, Atiewi S, Razaque A. Deep recurrent neural network for IoT intrusion detection system.[J] Simul Model Pract Theory. 2020 May 1;101:102031.

[39] Tobiyama S, Yamaguchi Y, Shimada H, Ikuse T, Yagi T. Malware Detection with Deep Neural Network Using Process Behavior. In: Proceedings - International Computer Software and Applications Conference.[J] IEEE Computer Society; 2016. p. 577–82.

[40] Sharif M, Lucas K, Bauer L, Reiter MK, Shintre S. Optimization-Guided Binary Diversification to Mislead Neural Networks for Malware Detection. Proc - 2019 IEEE Symp Secur Priv Work SPW 2019 [C]. 2019;6. Available from: https://github.com/drhyrum/gym-malware

[41] Sharif M, Lucas K, Bauer L, Reiter MK, Shintre S. Optimization-Guided Binary Diversification to Mislead Neural Networks for Malware Detection. [J]. 2019;6.

[42] Choi J, Shin D, Kim H, Seotis J, Hong JB. AMVG: Adaptive malware variant generation framework using machine learning. In: Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing, PRDC [C]. IEEE Computer Society; 2019. p. 246–55. Available from: https://ieeexplore.ieee.org/document/8952122/

[43] Sarker IH, Kayes ASMM, Badsha S, Alqahtani H, Watters P, Ng A. Cybersecurity data science: an overview from machine learning perspective. J Big Data [J]. 2020 Dec 1;7(1):41. Available from: https://link.springer.com/articles/10.1186/s40537-020-00318-5

[44] van der Aalst W, van der Aalst W. Data Science in Action. In: Process Mining [J]. Springer Berlin Heidelberg; 2016. p. 3–23. Available from: https://link.springer.com/chapter/10.1007/978-3-662-49851-4_1

[45] Dhar V. Data science and prediction. Commun ACM [C]. 2013 Dec 1;56(12):64–73. Available from: https://dl.acm.org/doi/abs/10.1145/2500499

[46] Data Science and Its Application in Cyber Security (Cyber Security Data Science) by Shiv Hari Tewari :: SSRN [J]. Available from: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3687251

[47] Sikos L, Choo K-K. Data Science in Cybersecurity and Cyberthreat Intelligence [J]. Sikos LF, Choo K-KR, editors. Springer. Cham: Springer International Publishing; 2019. (Intelligent Systems Reference Library; vol. 177). Available from: http://link.springer.com/10.1007/978-3-030-38788-4

[48] Yavanoglu O, Aydos M. A review on cyber security datasets for machine learning algorithms[J]. In: Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017. Institute of Electrical and Electronics Engineers Inc.; 2017. p. 2186–93.

[49] Bazrafshan Z, Hashemi H, Fard SMH, Hamzeh A. A survey on heuristic malware detection techniques. In: IKT 2013 - 2013 5th Conference on Information and Knowledge Technology [J]. IEEE; 2013. p. 113–20. Available from: http://ieeexplore.ieee.org/document/6620049/

[50] Schultz MG, Eskin E, Zadok F, Stolfo SJ. Data mining methods for detection of new malicious executables. In: Proceedings 2001 IEEE Symposium on Security and Privacy S&P 2001 [J]. IEEE Comput. Soc; 2000. p. 38–49. Available from: http://ieeexplore.ieee.org/document/924286/

[51] Sandbox | Kaspersky [ED/OL]. Available from: https://www.kaspersky.com/enterprise-security/wiki-section/products/sandbox

[52] Scott J. Signature Based Malware Detection is Dead. Cybersecurity Think Tank, Inst Crit Infrastruct Technol [R]. 2017 [cited 2021 Apr 30];(February). Available from: www.ICITForum.org

[53] Sahay SK, Sharma A, Rathore H. Evolution of Malware and Its Detection Techniques[J]. In: Advances in Intelligent Systems and Computing. 2020.

[54] Singh A. Portable Executable File Format. In: Identifying Malicious Code Through Reverse Engineering [J]. Boston, MA: Springer US; 2009. p. 1–15. Available from: https://link.springer.com/chapter/10.1007/978-0-387-89468-3_3

[55] Andersson H. application/vnd.microsoft.portable-executable [EB/OL]. IANA; 2015. Available from: https://www.iana.org/assignments/media-types/application/vnd.microsoft.portable-executable

[56] Devi D, Nandi S. PE File Features in Detection of Packed Executables. Int J Comput Theory Eng [J]. 2012;476–8. Available from: http://www.ijcte.org/papers/512-S10014.pdf

[57] Choi YS, Kim IK, Oh JT, Ryou JC. PE file header analysis-based packed PE file detection technique (PHAD).[C]. In: Proceedings - International Symposium on Computer Science and Its Applications, CSA 2008. 2008. p. 28–31.

[58] Zaidan AA, Zaidan BB OF. New technique of hidden data in PE-file with in unused area one. Int J Comput Electr Eng. 2009[J];(1793-8163.):1(5).

[59] Choi Y-S, Kim I-K, Oh J-T, Ryou J-C. Encoded Executable File Detection Technique via Executable File[J]. Int J Hybrid Inf Technol. 2009;2(2):25–36.

[60] Bridge K, Kennedy J, Batchelor D, Coulter D, Krell J, Kerdolph R, et al. PE Format - Win32 apps | Microsoft Docs [P]. Microsoft. 2021. Available from: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

[61] Malware Researcher’s Handbook (Demystifying PE File) - Infosec Resources [M]. Available from: https://resources.infosecinstitute.com/topic/2-malware-researchers-handbook-demystifying-pe-file/

[62] NIST. 1.1.1. What is EDA? Anal Explor Data Eda, [S]. 2015;(1977):2–3. Available from: https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm

[63] Prasad Patil. What is Exploratory Data Analysis? [P]. Towards Data Science. 2018. Available from: https://www.ibm.com/cloud/learn/exploratory-data-analysis

[64] Sun C. Feature Selection. In: Encyclopedia of Systems Biology [J]. New York, NY: Springer New York; 2013. p. 737–737. Available from: http://link.springer.com/10.1007/978-1-4419-9863-7_431

[65] Chandrashekar G, Sahin F. A survey on feature selection methods. Comput Electr Eng. 2014 Jan 1;40(1):16–28.

[66] How to Choose a Feature Selection Method For Machine Learning [EB/OL]. Available from: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

[67] Picard RR, Berk KN. Data Splitting. Am Stat [EB/OL]. 1990 May;44(2):140–7. Available from: http://www.tandfonline.com/doi/abs/10.1080/00031305.1990.10475704

[68] Training Parameters - Amazon Machine Learning [R]. Available from: https://docs.aws.amazon.com/machine-learning/latest/dg/training-parameters1.html#regularization

[69] Alpaydin Ethem. Introduction to Machine Learning - Ethem Alpaydin - Google Books [M]. Massachusetts Institute of Technology. 2020 [cited 2021 Feb 17]. 640 p. Available from: https://books.google.com.gh/books?hl=en&lr=&id=tZnSDwAAQBAJ&oi=fnd&pg=PR7&ots=F3SSaX4nEc&sig=lWutcsK7Re6_khNO3YmeCQ_F6Ac&redir_esc=y#v=onepage&q&f=false

[70] Smith R, Dullerud G, Rangan S, Poolla K. Model validation for dynamically uncertain systems. Math Model Syst [J]. 1997 Jan;3(1):43–58. Available from: https://www.tandfonline.com/action/journalInformation?journalCode=nmcm20

[71] Cross-Validation - Amazon Machine Learning [EB/OL]. Available from: https://docs.aws.amazon.com/machine-learning/latest/dg/cross-validation.html

[72] Model Fit: Underfitting vs. Overfitting - Amazon Machine Learning [EB/OL]. Available from: https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

[73] Odegua R. How to put machine learning models into production - Stack Overflow Blog [EB/OL]. [cited 2021 Mar 13]. Available from: https://stackoverflow.blog/2020/10/12/how-to-put-machine-learning-models-into-

[74] Bruzzone L, Persello C. A novel context-sensitive semisupervised SVM classifier robust to mislabeled rraining samples[J]. IEEE Trans Geosci Remote Sens. 2009 Jul;47(7):2142–54.

[75] Khammas BM, Monemi A, Bassi JS, Ismail I, Nor SM, Marsono MN. Feature selection and

machine learning classification for malware detection. J Teknol [J]. 2015 Nov 1;77(1):243–50. Available from: www.jurnalteknologi.utm.my

[76] Li W, Ge J, Dai G. Detecting Malware for Android Platform: An SVM-Based Approach[J]. In: Proceedings - 2nd IEEE International Conference on Cyber Security and Cloud Computing, CSCloud 2015 - IEEE International Symposium of Smart Cloud, IEEE SSC 2015. Institute of Electrical and Electronics Engineers Inc.; 2016. p. 464–9.

[77] Tang X, Cao R, Cheng J, Fan D, Tu W. DDoS attack detection method based on V-Support vector machine. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) [J]. Springer; 2019. p. 42–56. Available from: https://link.springer.com/chapter/10.1007/978-3-030-37352-8_4

[78] Kotpalliwar M V., Wajgi R. Classification of attacks using support vector machine (SVM) on KDDCUP'99 IDS database. In: Proceedings - 2015 5th International Conference on Communication Systems and Network Technologies, CSNT 2015. Institute of Electrical and Electronics Engineers Inc.; 2015. p. 987–90.

[79] Xie M, Hu J, Slay J. Evaluating host-based anomaly detection systems: Application of the one-class SVM algorithm to ADFA-LD[C]. In: 2014 11th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2014. Institute of Electrical and Electronics Engineers Inc.; 2014. p. 978–82.

[80] Gautam D, Tokekar V. Pattern Based Detection and Mitigation of DoS Attacks in MANET Using SVM-PSO[J]. In Springer, Cham; 2020. p. 176–84. Available from: https://link.springer.com/chapter/10.1007/978-3-030-44758-8_16

[81] Thakkar A, Lohiya R. Role of swarm and evolutionary algorithms for intrusion detection system: A survey. Swarm Evol Comput. 2020 Mar 1;53:100631.

[82] Saxena H, Richariya V. Intrusion Detection in KDD99 Dataset using SVM-PSO and Feature Reduction with Information Gain[J]. Int J Comput Appl. 2014;98(6):25–9.

[83] Chandrasekhar AM, Raghuveer K. Confederation of FCM clustering, ANN and SVM techniques to implement hybrid NIDS using corrected KDD cup 99 dataset.[C]. In: International Conference on Communication and Signal Processing, ICCSP 2014 - Proceedings. Institute of Electrical and Electronics Engineers Inc.; 2014. p. 672–6.

[84] Shapoorifard H, Shamsinejad P. Intrusion Detection using a Novel Hybrid Method Incorporating an Improved KNN. Int J Comput Appl. 2017;173(1):5–9.

[85] Vishwakarma S, Sharma V, Tiwari A. An Intrusion Detection System using KNN-ACO Algorithm[J]. Int J Comput Appl. 2017;171(10):18–23.

[86] Meng W, Li W, Kwok L-FF. Design of intelligent KNN-based alarm filter using knowledge-based alert verification in intrusion detection. Secur Commun Networks [J]. 2015 Dec

1;8(18):3883–95. Available from: http://doi.wiley.com/10.1002/sec.1307

[87] Dada EG. A Hybridized SVM-kNN-pdAPSO Approach to Intrusion Detection System [J]. Vol. 8, University of Maiduguri Faculty of Engineering Seminar Series. 2017. Available from: https://fardapaper.ir/mohavaha/uploads/2018/07/Fardapaper-A-Hybridized-SVM-kNN-pdAPSO-Approach-to-Intrusion-Detection-System.pdf

[88] Li W, Yi P, Wu Y, Pan L, Li J. A new intrusion detection system based on KNN classification algorithm in wireless sensor network.[J]. J Electr Comput Eng. 2014;2014.

[89] Lin WC, Ke SW, Tsai CF. CANN: An intrusion detection system based on combining cluster centers and nearest neighbors. Knowledge-Based Syst [J]. 2015 Apr 1;78(1):13–21. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0950705115000167

[90] Koc L, Mazzuchi TA, Sarkani S. A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier[J]. Expert Syst Appl. 2012 Dec 15;39(18):13492–500.

[91] Sarker IH, Abushark YB, Alsolami F, Khan AI. IntruDTree: A machine learning based cyber security intrusion detection model. Symmetry (Basel) [J]. 2020 May 1;12(5):754. Available from: www.mdpi.com/journal/symmetry

[92] Ingre B, Yadav A, Soni AK. Decision tree based intrusion detection system for NSL-KDD dataset. In: Smart Innovation, Systems and Technologies [J]. Springer Science and Business Media Deutschland GmbH; 2018. p. 207–18. Available from: https://link.springer.com/chapter/10.1007/978-3-319-63645-0_23

[93] Malik AJ, Khan FA. A hybrid technique using binary particle swarm optimization and decision tree pruning for network intrusion detection. Cluster Comput [J]. 2017 Jun 8;21(1):667–80. Available from: https://link.springer.com/article/10.1007/s10586-017-0971-8

[94] Relan NG, Patil DR. Implementation of network intrusion detection system using variant of decision tree algorithm[C]. In: 2015 International Conference on Nascent Technologies in the Engineering Field, ICNTE 2015 - Proceedings. Institute of Electrical and Electronics Engineers Inc.; 2015.

[95] Rai K, Syamala Devi M, Syamala Devi Professor M, Guleria A. Decision Tree Based Algorithm for Intrusion Detection multiagent system for management and evaluation of examination system View project Multiagent System for Management and Evaluation of Computer Science Examinations View project Decision Tree Based Algo. researchgate.net [J]. 2016; Available from: https://www.proftriumph.com

[96] Puthran S, Shah K. Intrusion detection using improved decision tree algorithm with binary and quad split[J]. In: Communications in Computer and Information Science. Springer Verlag; 2016. p. 427–38.

[97] Ahmad B, Jian W, Shafiq M. Intrusion Detection by Using Hybrid of Decision Tree and K-Nearest Neighbor. Int J Hybrid Inf Technol [J]. 2016;9(12):201–8. Available from: https://www.researchgate.net/publication/282326950

[98] Ullah F, Javaid Q, Salam A, Ahmad M, Sarwar N, Shah D, et al. Modified Decision Tree Technique for Ransomware Detection at Runtime through API Calls[J]. Sci Program. 2020;2020.

[99] Tajbakhsh A, Rahmati M, Mirzaei A. Intrusion detection using fuzzy association rules. Appl Soft Comput J [J]. 2009;9(2):462–9. Available from: https://www.sciencedirect.com/science/article/pii/S1568494608000975

[100] Mitchell R, Chen IR. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. IEEE Trans Dependable Secur Comput [J]. 2015;12(1):16–30. Available from: https://ieeexplore.ieee.org/abstract/document/6774867/

[101] Alazab M, Venkataraman S, Watters P. Towards understanding malware behaviour by the extraction of API calls. In: Proceedings - 2nd Cybercrime and Trustworthy Computing Workshop, CTC 2010 [J]. 2010. p. 52–9. Available from: https://www.researchgate.net/publication/230554531

[102] Yuan Y, Kaklamanos G, Hogrefe D. A novel semi-supervised adaboost technique for network anomaly detection. In: MSWiM 2016 - Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems [C]. Association for Computing Machinery, Inc; 2016. p. 111–4. Available from: http://dx.doi.org/10.1145/2988287.2989177

[103] Aslahi-Shahri BM, Rahmani R, Chizari M, Maralani A, Eslami M, Golkar MJ, et al. A hybrid method consisting of GA and SVM for intrusion detection system[J]. Neural Comput Appl. 2016 Aug 1;27(6):1669–76.

[104] Kim JJ, Kim JJ, Thu HLT, Kim H. Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. In: 2016 International Conference on Platform Technology and Service, PlatCon 2016 - Proceedings [J]. Institute of Electrical and Electronics Engineers Inc.; 2016. Available from: https://www.researchgate.net/publication/301583622

[105] Kolosnjaji B, Zarras A, Webster G, Eckert C. Deep learning for classification of malware system call sequences[J]. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag; 2016. p. 137–49.

[106] Alauthman M, Aslam N, Al-kasassbeh M, Khan S, Al-Qerem A, Raymond Choo KK. An efficient reinforcement learning-based Botnet detection approach. J Netw Comput Appl [J]. 2020;150. Available from: https://doi.org/10.1016/j.jnca.2019.102479

[107] Blanco R, Cilla JJ, Briongos S, Malagón P, Moya JM. Applying Cost-Sensitive Classifiers with Reinforcement Learning to IDS[J]. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag; 2018. p. 531–8.

[108] Lopez-Martin M, Carro B, Sanchez-Esguevillas A. Application of deep reinforcement learning to intrusion detection for supervised problems. Expert Syst Appl [J]. 2020;141. Available from: https://www.sciencedirect.com/science/article/pii/S0957417419306815

[109] Yuval tisf Nativ. theZoo. Available from: https://github.com/ytisf/theZoo

[110] Ronen R, Radu M, Feuerstein C, Yom-Tov E, Ahmadi M. Microsoft malware classification challenge. arXiv. 2018. Available from: https://www.kaggle.com/c/malware-classification

[111] Microsoft. Strings - Windows Sysinternals | Microsoft Docs [EB/OL]. docs.microsoft.com. 2020. Available from: https://docs.microsoft.com/en-us/sysinternals/downloads/strings

[112] Ramos J. Using TF-IDF to Determine Word Relevance in Document Queries. Proc first Instr Conf Mach Learn. 2003;242(1):29–48.

[113] Instr Conf Mach Learn. 2003;242(1):29–48. Broder AZ. Syntactic clustering of the Web. Comput Networks. 1997 Sep 1;29(8–13):1157–66

[114] Zipf Distribution - an overview | ScienceDirect Topics. Available from: https://www.sciencedirect.com/topics/computer-science/zipf-distribution

[115] Wu D, Fang B, Wang J, Liu Q, Cui X. Evading Machine Learning Botnet Detection Models via Deep Reinforcement Learning[J]. In: IEEE International Conference on Communications. Institute of Electrical and Electronics Engineers Inc.; 2019.

Main research achievements during the period of studying for master's degree

1. Papers

[1] Zhao, M., Wang, Q., Ning, J. et al. A region fusion based split Bregman method for TV Denoising algorithm. Multimedia Tools Appl 80, 15875– 15900 (2021). https://doi.org/10.1007/s11042-020-10407-5

[2] M. Zhao, Q. Wang, A. N. Muniru, J. Ning, P. Li and B. Li, "Numerical Calculation of Partial Differential Equation Deduction in Adaptive Total Variation Image Denoising," 2019 12th International Congress on Image

2. Awards During Master’s Degree

[1] 2020-2021 Xi’an International Entrepreneurship Competition 2020.

[2] 2019-2020 One-Belt-One Road International Student’s Entrepreneurship Competition-2019. Xi’an China.