



Islamic University of Technology (IUT)

Department of Computer Science and Engineering (CSE)

## **A Study of Permission-based Malware Detection Using Machine Learning**

Authors

**Md. Rafid Islam - 170042009**

**Ratun Rahman - 170042011**

**Akib Ahmed - 170042013**

Supervisors

Dr. Kamrul Hasan

Professor, Head ICT Center

Department of CSE

Dr. Hasan Mahmud

Assistant Professor

Department of CSE

**A thesis submitted to the Department of CSE  
in partial fulfillment of the requirements for the degree of  
B.Sc. in Software Engineering**

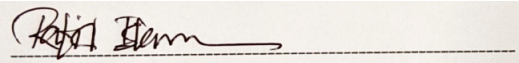
**Academic Year: 2020-21**

**April - 2022**

# Declaration of Authorship

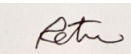
This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Md. Rafid Islam, Ratun Rahman and Akib Ahmed under the supervision of Dr. Kamrul Hasan, Professor, Head of ICT Center, and Dr. Hasan Mahmud, Assistant Professor, Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

*Authors:*



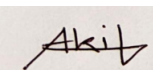
-----  
Md. Rafid Islam

Student ID - 170042009



-----  
Ratun Rahman

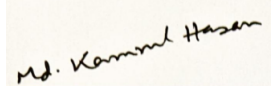
Student ID - 170042011



-----  
Akib Ahmed

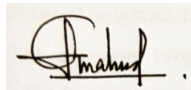
Student ID - 170042013

*Supervisor:*



---

Dr. Kamrul Hasan  
Professor, Head of ICT Center  
Department of Computer Science and Engineering  
Islamic University of Technology (IUT)



---

Dr. Hasan Mahmud  
Assistant Professor  
Department of Computer Science and Engineering  
Islamic University of Technology (IUT)

# Acknowledgement

We would like to express our grateful appreciation for **Dr. Kamrul Hasan and Dr. Hasan Mahmud**, Department of Computer Science & Engineering, IUT for being our adviser and mentor. Their motivation, suggestions and insights for this research have been invaluable. Without their support and proper guidance this research would never have been possible. Their valuable opinion, time and input provided throughout the thesis work, from first phase of thesis topics introduction, subject selection, proposing algorithm, modification till the project implementation and finalization which helped us to do our thesis work in proper way. We are really grateful to him.

We are also grateful to Almighty ALLAH and our parents.

# Abstract

Malware is becoming more prevalent, and several threat categories have risen dramatically in recent years. This paper provides a bird's-eye view of the world of malware analysis. It also presents a brief review of malware analysis approaches, common detection types, and some basic preventive strategies from various angles. An experiment has been done to show the influence of human factors on people. This study shows that most people are more likely to fall victim to a malware attack if that seems to come from a reliable source or person. The efficiency of five different machine learning methods (Naive Bayes, K-Nearest Neighbor, Decision Tree, Random Forest, Decision Forest) combined with features picked from the retrieval of Android permissions to categorize applications as harmful or benign is investigated in this study. On a test set consisting of 1,168 samples (each consisting of 948 features), produce accuracy rates above 80% (Except Naive Bayes Algorithm with 65% accuracy). Of the considered algorithms TensorFlow Decision Forest performed the best with an accuracy of 90%.

## Keywords

Malware, malware analysis, malware detection, malware prevention, decision forest

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>Literature Review</b>	<b>10</b>
2.1	Malware Overview . . . . .	10
2.1.1	Virus . . . . .	10
2.1.2	Worm . . . . .	11
2.1.3	Trojan horse . . . . .	12
2.1.4	Spyware . . . . .	12
2.1.5	Adware . . . . .	13
2.1.6	Ransomware . . . . .	13
2.2	Evolution of Malware . . . . .	14
2.3	Recent Threat Analysis . . . . .	15
2.4	Machine Learning Algorithms . . . . .	22
2.4.1	Naive Bayes Algorithm . . . . .	22
2.4.2	K-Nearest Neighbors . . . . .	23
2.4.3	Decision Tree Algorithm . . . . .	23
2.4.4	Random Forests Algorithm . . . . .	24
2.4.5	TensorFlow Decision Forests Algorithm . . . . .	24
<b>3</b>	<b>Malware Distribution on Different OS</b>	<b>26</b>
3.1	Malware Attack on Different OS . . . . .	26
3.1.1	Windows . . . . .	26
3.2	Why Linux is Secured . . . . .	27
3.3	Why Android is Vulnerable . . . . .	29
<b>4</b>	<b>Malware Analysis</b>	<b>32</b>
4.1	Malware Analysis Techniques . . . . .	32
4.1.1	Static Analysis . . . . .	32
4.1.2	Dynamic Analysis . . . . .	33

4.1.3	Hybrid Analysis . . . . .	34
4.2	Malware Detection . . . . .	35
4.2.1	Signature-based detection . . . . .	35
4.2.2	Heuristic-based detection . . . . .	36
4.2.3	Specification-based detection . . . . .	36
4.2.4	Machine learning-based detection . . . . .	37
4.2.5	Cloud-based detection . . . . .	37
4.3	Malware Prevention . . . . .	38
4.3.1	Policy . . . . .	38
4.3.2	Awareness . . . . .	38
4.3.3	Patch Management . . . . .	39
4.3.4	Least Privilege . . . . .	39
4.3.5	Other host hardening measures . . . . .	40
4.3.6	Threat mitigation . . . . .	40
4.3.7	Antivirus Software . . . . .	40
4.3.8	Spyware detection and removal utilities . . . . .	41
4.3.9	Intrusion prevention systems . . . . .	41
4.3.10	TPM and Secure Boot . . . . .	41
4.3.11	Firewall and routers . . . . .	42
<b>5</b>	<b>Related Works</b>	<b>43</b>
<b>6</b>	<b>Proposed Approach</b>	<b>45</b>
6.1	Methodology . . . . .	45
6.2	Environment . . . . .	46
6.3	Data Collection . . . . .	47
6.4	Design and Testing . . . . .	47
<b>7</b>	<b>Result Analysis</b>	<b>50</b>
<b>8</b>	<b>Limitations and Challenges of The Conducted Work</b>	<b>70</b>
8.1	Limitations . . . . .	70

8.2 Challenges . . . . .	70
<b>9 Conclusion and Future Work</b>	<b>72</b>



## List of Figures

1	Virus . . . . .	11
2	Worm . . . . .	11
3	Trojan Horse . . . . .	12
4	Spyware . . . . .	13
5	Adware . . . . .	13
6	Ransomware . . . . .	14
7	Top countries with most malware detections . . . . .	16
8	Top sectors with most malicious software detection . . . . .	16
9	Top 10 attack vectors . . . . .	17
10	New ransomware in recent year . . . . .	18
11	New MacOS malware in recent year . . . . .	18
12	New linux malware in recent year . . . . .	19
13	New iOS malware in recent year . . . . .	19
14	New mobile malware in recent year . . . . .	20
15	New coin miner malware in recent year . . . . .	20
16	New IoT malware in recent year . . . . .	21
17	New powershell malware in recent year . . . . .	21
18	New malware in recent year . . . . .	22
19	Total malware in recent year . . . . .	22
20	Distribution of malware attacks in different OS . . . . .	26
21	Distribution of malware under windows . . . . .	27
22	Linux kernel development . . . . .	28
23	Most vulnerable operating system . . . . .	30
24	Android architecture . . . . .	31
25	Android vulnerable in each layer . . . . .	32
26	Malware detection techniques . . . . .	35
27	Workflow of the Paper . . . . .	46
28	Questionnaire from the Experiment . . . . .	51

29	A simple ransomware example . . . . .	52
30	Confidence Measurements of Label . . . . .	53
31	Comparison of Malicious Actions Between Two Malwares . . . . .	53
32	Permission Analysis from the Detailed Report . . . . .	55
33	Summary report of the Application . . . . .	56
34	Compiler, Manipulator Details of Various Applications . . . . .	57
35	Properties Analysis Report of Application . . . . .	58
36	User-Interface Transition Diagram of Customized Application. . . . .	60
37	Pictorial Overview of the Permission List Gathered. . . . .	64
38	Top 10 Permissions Used (Benign vs Malware) . . . . .	65
39	Permissions (Malware vs Benign) . . . . .	67
40	Model for TensorFlow Serving. . . . .	68
41	Accuracy and Data Loss Graph of TensorFlow Decision Forest Algorithm . . . . .	69

## List of Tables

1	Structure data table . . . . .	25
2	Structure data using decision forest . . . . .	25
3	Difference between static and dynamic analysis . . . . .	33
4	Advantages and disadvantages of various detection techniques . . . .	36
5	Analysis of Packages Reported as Malicious (Android 4.4, API 19)	61
6	Analysis of Packages Reported as Malicious (Android 8.1, API 27) .	62
7	Partial Analysis of Packages Reported as Malicious (Android 10, API 29) . . . . .	63
8	Comparative Results of Malicious Packages in Different Version . .	63
9	Different machine learning algorithm accuracy . . . . .	68

# 1 Introduction

Malware is on the rise. According to the ‘McAfee Labs Threat Report 2021’, the years 2019 and 2020 have seen a significant increase in several threat categories, in which malware took the first place. Understanding how different forms of malware work and how to identify them is critical for properly securing networks [4]. To carry out an attack, malicious hackers frequently mix multiple forms of malware. Security professionals must be aware of the malware types that are frequently combined, as well as how to build resilience against these threats [6].

Identifying malware is a core objective to defend against malware attacking a system or removing malware from a compromised software system. We need to check some characteristics such as how malware functions, its performance, and its creating purposes. Therefore, the first step in detecting malware is malware analysis [1].

Malware analysis focuses on observing how a suspicious file, URL, or software behaves in a particular system and what the purpose is. The procedure’s output helps in the detection and also the quick response of any dangerous outcome. It can significantly support incident responders and security analysts [7].

There have been some major malware invasions and security threats in recent years. The Estonian cyber attack that took place in 2007 targeted many institutions in the country, including government establishments. The data breach that took place at Yahoo in 2014 has been a major setback for the company. The perpetrators stole data from more than 500 million users. Pegasus first came into spotlight in 2016 as the dangerous spyware and recently it resurfaced again in 2021 and it can infiltrate any device and network without leaving a trace. Still there is no efficient way to detect or prevent it that has been disclosed to the world. Ransomware has proven to be a headache for many people. People are getting scammed and losing money because of ransomware. In 2021, Acer got invaded by ransomware from the REvil hacker group. CDPprojekt Red, a well known publisher

name in the gaming industry fell into the trap of ransomware as well in 2021 and refused to pay ransom as they already had backups. But not everyone can avoid this looming threat of ransomware and many end up paying in the end as they have no other way out. Ensuring safety of users is of utmost importance. People have their personal data stored in their smart devices or cloud storages and sometimes this information is crucial to their life and wellbeing. So, detecting malware effectively and preventing security threats should be given the highest priority.

Malware creators make it a priority to make their creations undetectable. As time went by, malware authors updated their techniques, and malware has become more difficult to detect. Malware has become more sophisticated and complex. There are limitations when it comes to malware detection techniques, and there is room for improvement as well. Combining machine learning methods with already established detection techniques can have a significant impact on the field of malware detection.

Malware that is already known or has been discovered before can be detected using signature-based malware detection. But the major flaw of signature-based malware detection is that this particular method will render itself useless when there is malware that has not been discovered yet. Also, polymorphic malware can change its signature. The solution to this is heuristic-based malware detection, which is effective against both discovered and undiscovered malware. But this method has its downside as well, and that would be a high rate of false positives and negatives. So a combination of techniques is needed to battle the ever growing malware threat, and that's why machine learning is being used in these cases to get better results.

This paper takes a dive into malware analysis, detection techniques, some prevention methods and then presents an experiment of five machine learning algorithms being used in permission based malware detection and the results of that experiment.

Section 2 gives a literature review of the world of malware. The distribution of

malware on various operating systems is discussed in section 3. Analysis of malware is given in section 4. It covers a wider topic consisting of malware detection and prevention techniques. Section 5 gives a review of related works. The proposed approach for this paper is mentioned in section 6, with the result analysis being done in section 7. Section 8 indicates the limitations and challenges faced in conducting the work. Finally, the conclusion and future work finish the paper in section 9.

## 2 Literature Review

Malware is a short term that is composed of the two words "malicious" and "software", which encompasses all types of software or programs designed to carry out detrimental actions on a system [2]. Viruses, trojan horses, worms, ransomware, adware, rootkits, wipers, logic bombs, bots, or any kind of malicious code or program that infiltrates a computer with the intention of wreaking havoc can be classified as malicious software or malware. According to Microsoft, "[malware] is a catch-all term to refer to any software designed to cause damage to a single computer, server, or computer network." Malware is destructive or manipulative software that interferes with the usual operation of an electronic device. Malware can leave a ruinous impact on personal computers, servers, tablets, smartphones, and any other device that possesses computing proficiency [3].

There is one thing that needs to be clarified, and it is the particular process or technique that is followed that ends up making malware is not at fault, nor can it be used as an identifier of malware [5]. It is the intention behind the creation that is scrutinized when identifying a program or software as malware. In short, the intention of the maker is at stake here, not the making procedure or the features.

### 2.1 Malware Overview

Malware is classified into different categories based on how it attacks and spreads. Here are six of the most common varieties of malware, along with their features [8]. Each of them are represented with symbolic images from figure- 1 to figure- 6.

#### 2.1.1 Virus

Virus is a malware that can spread and duplicate itself by piggybacking on normal application code. Viruses try to harm a particular device by inserting corrupted data, deleting hard drive information, or auto shutdown the system. They can also steal valuable data, harm the networks, create botnets, pirate, generate self-advertising, and many more [9]. Computer viruses are dangerous because they

can spread very quickly unlike others [10].



Figure 1: Virus

### 2.1.2 Worm

Worm is a self-replicating malware that can spread without human interactions. The key difference between a virus and a worm is, that the worm can clone itself without any help. Viruses need the user to do action, such as downloading a file or opening an email, whereas worms do not. Worms can get access to a system in a variety of methods, including software backdoors, other computers on the same network, operating system vulnerabilities, flash drives, and so on [13].

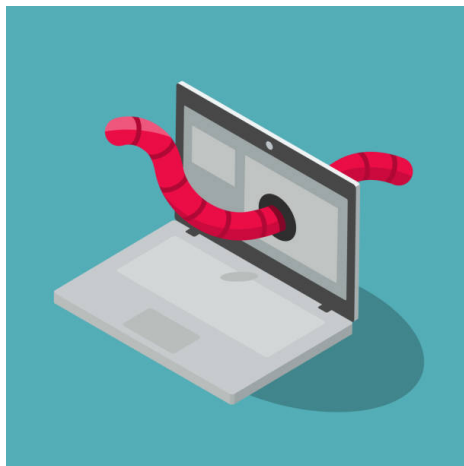


Figure 2: Worm



### 2.1.3 Trojan horse

Trojans are a type of malicious software that looks and behaves the appearance and behavior of legitimate software while containing malicious codes inside. For example, a Trojan might be an email advising a user to update his or her anti-virus software. As soon as the user hits the download button, it will be too late and their PC will be infected [15]. Trojan itself cannot duplicate itself and reproduce. However, mixing with a worm or virus, it can cause serious damage to a system or user.

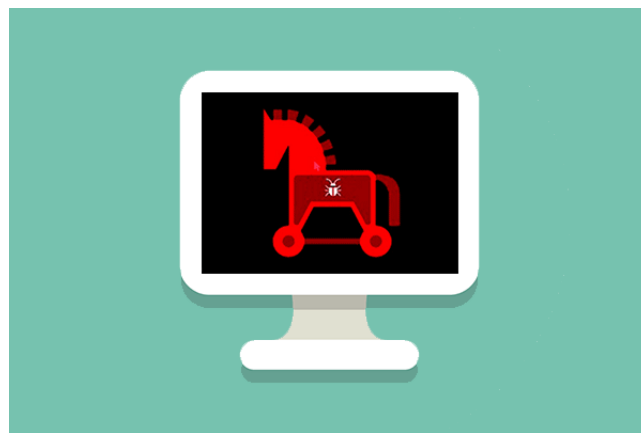


Figure 3: Trojan Horse

### 2.1.4 Spyware

A spyware is a type of application that enables hackers to monitor what you're doing on your device or what's running on your system. Hackers may easily steal your personal information by monitoring your activities, capturing personal information, passwords, monetary transactions, and more. Furthermore, this malware was clearly utilized to track your browsing behavior, including which sites you visited and when [17].



Figure 4: Spyware

### 2.1.5 Adware

Adware is a malware that puts unwanted advertising on screen. Those advertisements can be really aggressive at times as well. The purpose of Adware would be to collect data, redirect users to various advertising sites, and tinker with the browser settings, which includes changing the default browser, search settings, and homepage. However, there is some legit adware that will ask for permission before collecting information [17].



Figure 5: Adware

### 2.1.6 Ransomware

Ransomware is malicious software that restricts access to a device and demands some sort of resource for recovering the device. It either encrypts the data or perpetually blocks access. Then it displays notifications to the user for payment

to unlock. After payment, the attacker can unlock the device or give the key or password to the user to unlock it. Ransomware is the most common attack pattern in the 21st century, growing at an alarming rate over the years [19].



Figure 6: Ransomware

## 2.2 Evolution of Malware

Initially, what was created to carry out experiments or tests and pranks later evolved and emerged as a tool for cybervandalism. In the modern age of information technology and the internet, malware is a constant threat looming over our digital devices, but it was not so imposing in its rudimentary stage. Below, there is a representation of malware hallmarks throughout the years.

**Malware in the 1970s:** Authored by Bob Thomas in 1971, Creeper was an experimental self-replicating program, and it would be classified as a worm by today's standards. In 1974, Wabbit was a self-replicating program that made many copies of itself on a computer until it slowed the system down to the point where it crashed. The first Trojan, dubbed "ANIMAL", was created by Autodesk founder John Walker in 1975.

**The 1980s and 1990s:** The first serious Windows PC virus, called "Brain," wasn't released until 1986. Brain, like previous viruses, was mostly innocuous, albeit it did slow floppy disks to a halt and consume a significant amount of RAM [11].

As computer networks continued to be adopted and expanded throughout the 1990s, malware propagation grew faster, and hence volume rose. Particular types of malware flourished as technologies became more standardized.

Malware in the Early 2000s: SQL Slammer, which was created in 2003, spread so quickly that it crippled the internet just under 30 minutes after it was released [12]. The Cabir virus came out in 2004 and is regarded as the first mobile phone virus [14]. Then there was Zeus in 2007, capable of obtaining the victim's banking information [16].

Malware Since 2010: During this decade, malware developed new distribution channels like social media, IoT, and cryptocurrency. ZeroAccess, a Microsoft-specific Trojan horse, used botnets to distribute malware onto PCs. Regin, a Trojan horse alleged to have been built for espionage and mass surveillance purposes in the US and the UK, was discovered in 2014. Cerber was one of the most widespread crypto-malware threats in 2016, and Microsoft discovered more enterprise PCs infected with Cerber than any other in the ransomware sphere. Thanatos was the first ransomware to take Bitcoin payments in 2018, as cryptocurrency became well known [18].

### **2.3 Recent Threat Analysis**

We have seen a rise in malware attacks in recent years as they are evolving constantly and proving to be a constant threat. We'll take a look at the top countries that have been infected with malware attacks and which sectors are most prevalent for malware attacks. According to MacAfee, there have been 16,436,403 malicious detections, 8,637 distinct hashes, and 10,008 unique organizations from the 3rd to 4th quarter of 2020. Figure- 7 represents a bar-graph showing the country-wise malicious software detection rates in comparison.

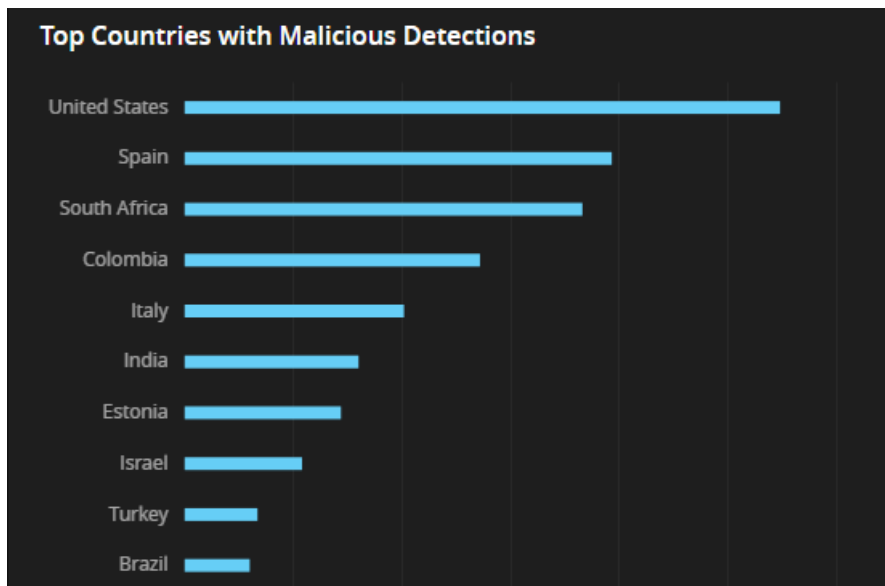


Figure 7: Top countries with most malware detections

The USA tops the list with the most malware detections (1,095,361), followed by Spain (786,021), South Africa (731,985), Colombia (543,754), Italy (403,981), India (320,443), Estonia (288,155), Israel (216,609), Turkey (134,665) and Brazil (120,426).

Figure- 8 depicts the top sectors where malicious software is detected the most.

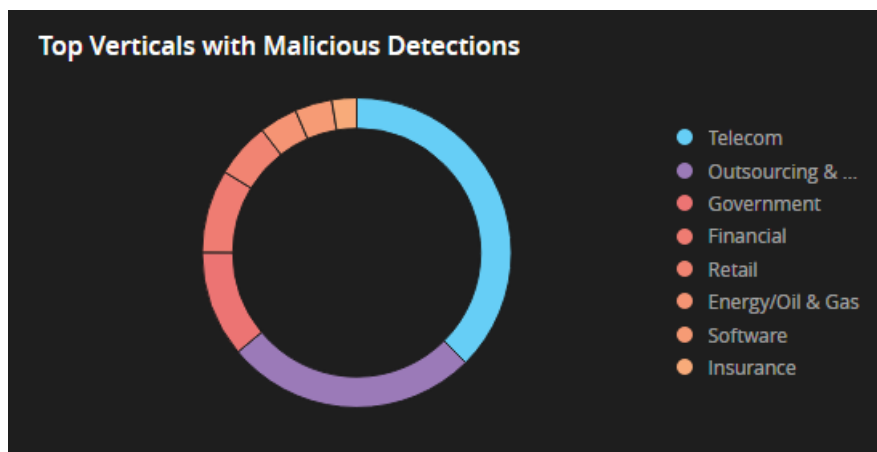


Figure 8: Top sectors with most malicious software detection

The telecom business has been plagued by malware more than any other sector, according to a McAfee analysis. Telecom sector tops the list with the most malware detections (37.6%), followed by Outsourcing (26.4%), Government (11.1%), Financial (8.7%), Retail (5.8%), Energy (4.0%), Software (3.9%), and Insurance (2.6%).

The top ten attack vectors are depicted in Figure- 9.

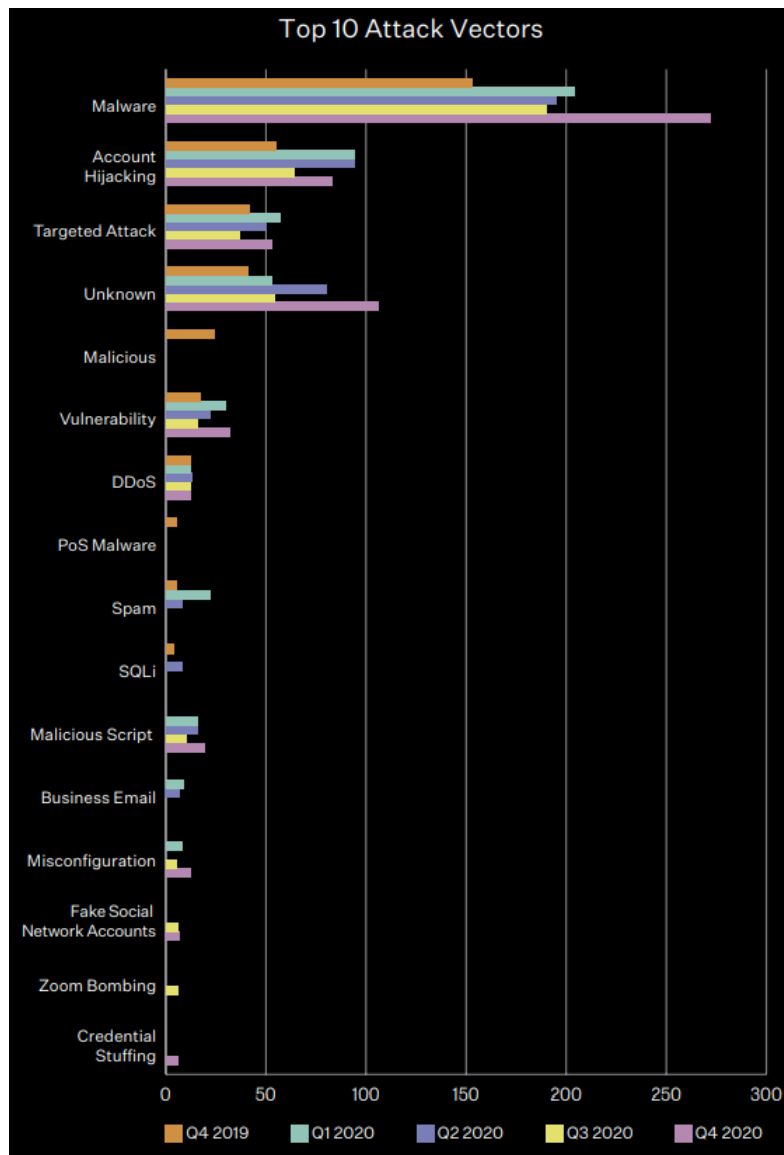


Figure 9: Top 10 attack vectors

Amongst all the attack vectors malware already topped the list in the 4th quarter of 2019. It increased in the 1st quarter of 2020 and kept decreasing until the 4th quarter of 2020, then it increased again vastly and it is expected to rise as the days go by.

Figures- 10 through 19 depict an overall situation of several malware variants in recent years.

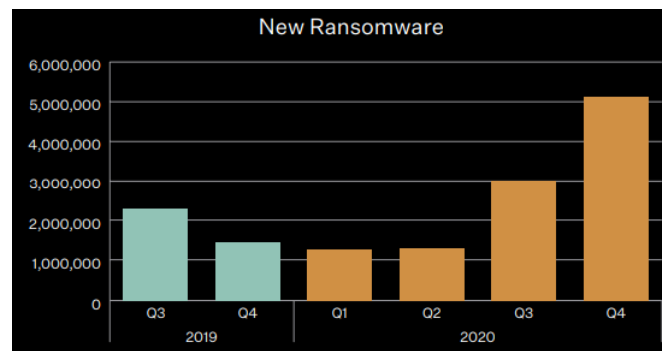


Figure 10: New ransomware in recent year

Ransomware has taken the world by surprise. Ransomware attacks are continuously increasing and as can be seen from the graph they kept rising throughout 2020 and reached their peak in the last quarter of the same year.

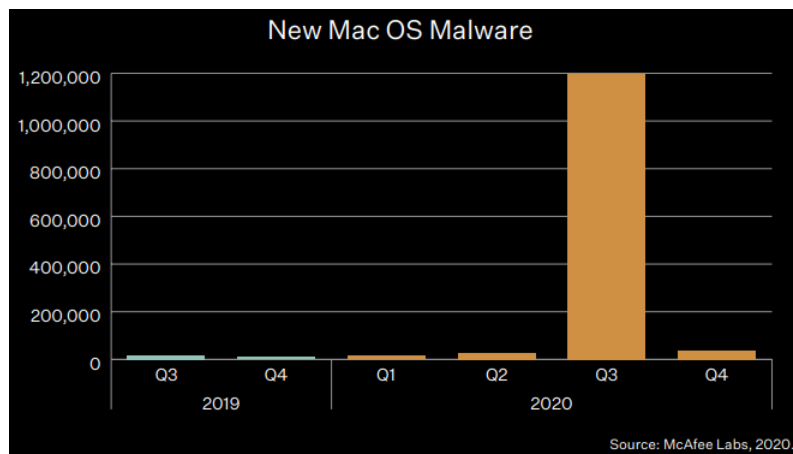


Figure 11: New MacOS malware in recent year

Mac has been penetrated by new malware at a lower rate than other operating systems. In the third quarter of 2020, new Mac OS Malware increased dramatically due to EvilQuest ransomware but quickly came down to the usual normal level in the last quarter of 2020.

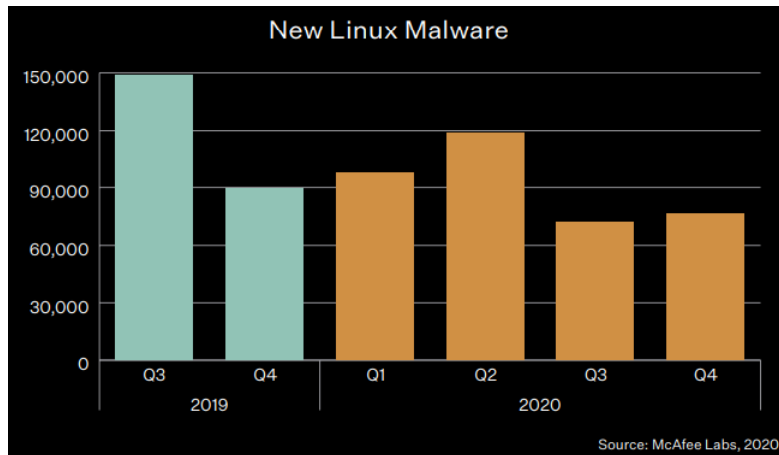


Figure 12: New linux malware in recent year

New Linux malware increased 6% from Q3 to Q4 but compared to the previous year it is still less. In the 1st quarter of 2019 it was at its peak.

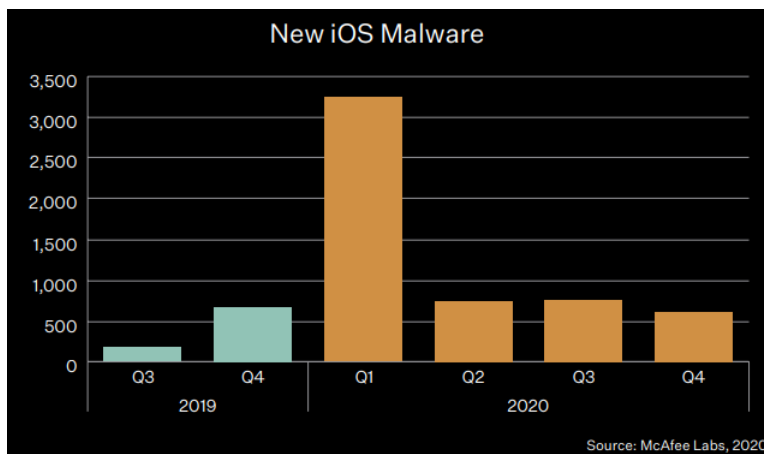


Figure 13: New iOS malware in recent year

iOS Malware exploded in the first quarter of 2020 and then came down quickly in the second quarter and remained normal throughout the rest of the year.



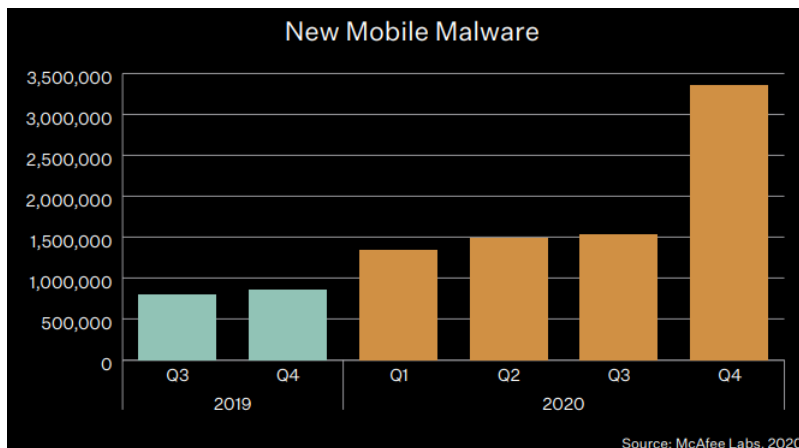


Figure 14: New mobile malware in recent year

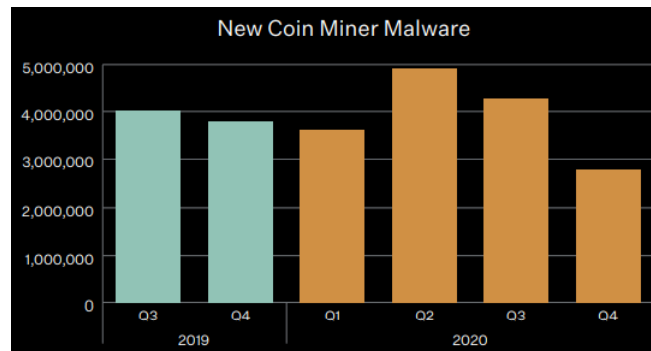


Figure 15: New coin miner malware in recent year

Mobile malware grew 118% from Q3 to Q4 driven by SMS Reg.

A Coinminer is a malicious program that mines for coins using the victim's computing resources, such as CPU and RAM, without the consent of the victim. Monero or Zcash are examples of Coinminer. Coinminer malware was on the rise from the 1st quarter till the 3rd quarter of 2020 and then it decreased in quantity in the 4th quarter of the same year.

IoT malware, as the name suggests, infiltrates an internet of things system. One of the characteristics of this type of malware is that it is utilized to deploy DDoS attacks. Mirai is an example of malware that has been used for this purpose. IoT malware has been through ups and downs throughout 2019 and 2020.

Powershell is generally used by Fileless malware to carry out an invasion on a system, leaving behind no signs of it. It can be very difficult to figure out if

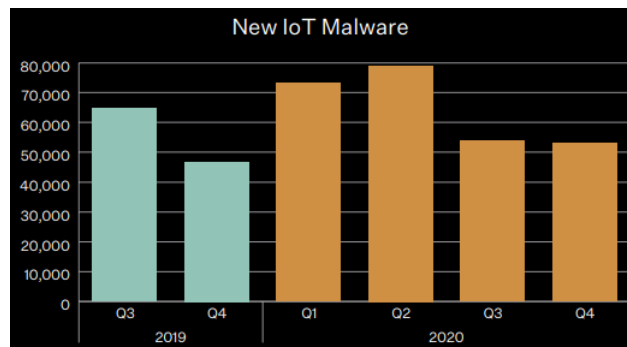


Figure 16: New IoT malware in recent year

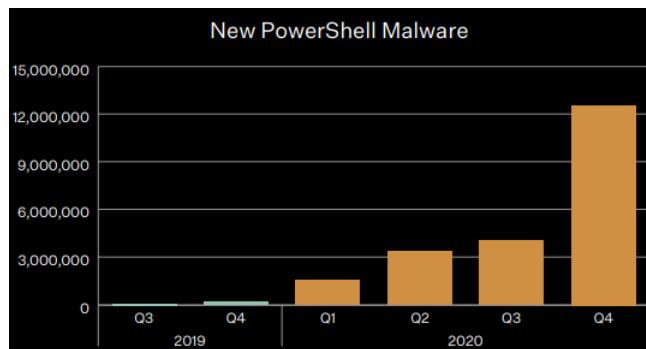


Figure 17: New powershell malware in recent year

a system has been infiltrated or not and this is referred to as a "zero-footprint attack". Powershell malware has been on the rise and it reached a new height in the last quarter of 2020.

New unique malware attacks has been on the rise throughout 2020. It decreased in the first quarter of 2020 meaning more unique malware came out in the final quarter of 2019. Then again in the final quarter of 2020, it surpassed the previous year's numbers.

There may have been ups and downs in the unique malware number but the total number of malware attacks kept increasing throughout 2019 and 2020 and it is expected to increase in the future as well.

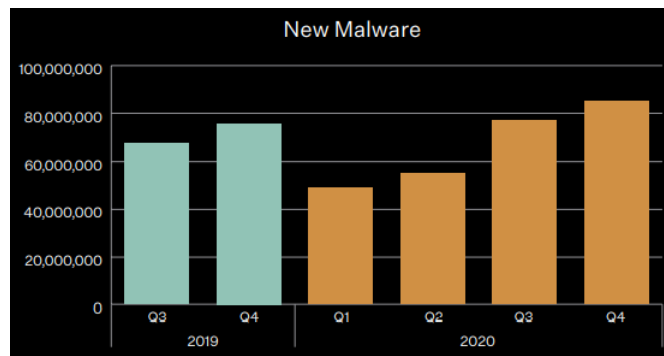


Figure 18: New malware in recent year

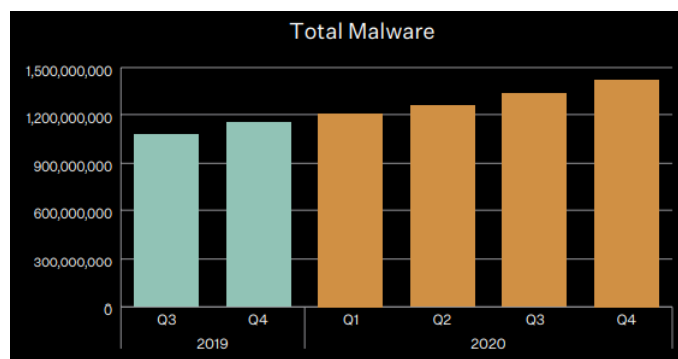


Figure 19: Total malware in recent year

## 2.4 Machine Learning Algorithms

### 2.4.1 Naive Bayes Algorithm

Naive Bayes is a probability-based algorithm focusing on simple calculations using the Bayes theorem. However, it is not a single algorithm, but a classification technique focusing on a group of independent variables compared with other objects that are not related because of that combination of values from the variables. For example, a chicken is an animal that has 2 legs one head and is around 27 inches long. While there can be more animals that match one or more properties with chicken, not all the properties match altogether. This is how we can differentiate chicken from all other animals group and find out a probability of an animal being a chicken using these variables and that's why it is known as naive.

As we can see, this algorithm simply uses some variables and probability algo-

gorithms to find the decisions. Therefore, it has many limitations. So it is mainly used on classifying objects for a very long dataset. As it is simple and easy to calculate, naive bayes can outperform other machine learning algorithms for huge datasets in terms of time, especially the real-time applications.

### **2.4.2 K-Nearest Neighbors**

K-Nearest Neighbours also known as KNN is a non-parametric algorithm that is based on Supervised learning techniques. In other words, it is used to measure the similarities between two objects and determine which is more likely to be the classifier. For example, if we have a tiger and two existing categories of chicken and cat, we want to know where should we put the tiger. To solve this issue, first, we compare the tiger with the chicken and find their similarities and differences. We applied the same between tiger and cat as well and then by judging the similarity, we figure out the group that it likely belongs to.

It is also used for the classification problem, the same as naive bayes. However, it does not make any assumption about the existing datasets, but stores them and will run when it times for classification.

### **2.4.3 Decision Tree Algorithm**

The decision tree algorithm uses a graph with a root determining the start and leaves to find out the probability of a particular event occurring among all the possible solutions. This technique is widely used in both classification and regression problems. For example in weather forecasting, we can use a decision tree algorithm to find out the probability of the next day's weather. By considering the past day's structure as a root and then growing the tree accordingly, we can find out a relative probability percentage and use the value to find out future values using the percentages.

So, the decision tree simply asks a boolean question that can be answered using yes/no and keep spreading the trees for all possibilities using a graphical representation. So when the decision is comparatively easy to take and every path can

be used as node as its parent, we can use a decision tree. It is fast, simple, and can make a longer prediction.

#### **2.4.4 Random Forests Algorithm**

Similar to the decision tree algorithm, the random forest uses a tree for determining all the possible solutions. However, it takes a majority vote into consideration for the decision problems and the average vote for the regression problem. So it is used both in classification and regression problems. For example, someone has gone to a local restaurant to eat but he has no idea which dish is good. So he asks his friends and neighbors who have been there before for recommendations using several questions. After consulting, he finally makes a decision based on the report.

However, this algorithm is slower than the decision tree algorithm as it takes time for observation. However, it does not require a specific formula to calculate and as a result, can be used to calculate non-boolean questions as well. It is widely used in for its efficiency because it can calculate the continuous number and also can calculate with having some missing values. So in real life where some of the questions are hard to answer with boolean and may exist missing information or can not be fit into an equation, the random forest algorithm is used.

#### **2.4.5 TensorFlow Decision Forests Algorithm**

Decision forest is a family of machine learning models and is easier to use than neural networks. The tree asks a series of simple boolean questions to classify an item from a dataset. The decision forest algorithm uses keras, an API for neural networks to experiment with different kinds of models to figure out which is the best for a particular data.

The best type of model to use depends on the types of data. Structured data is where a tree is the best to use. Structured data can be represented as data that can fit into a CSV file. This concept is explained with an example:

Example	Feather	Can Fly?	Label
1	yes	yes	Eagle
2	yes	no	Chicken
3	no	no	Cat

Table 1: Structure data table

Here in table- 1, while training a decision forest to classify a data, the features become the questions in the tree. The features described the understandable concept that can be easily calculated or observed and can be used as an independent variable to separate different kinds of objects. The last column (in table- 2) represents the answer. So our objective is to find the label as soon as possible. The decision forest tree uses labels as the ranking and uses that to separate further groups necessary. So decision forest is easy to use, interpretable, and powerful.

Example	Height	Weight	Group	Label
1	1.7 m	75 kg	Mammals	1
2	0.5 m	2 kg	Birds	2
3	1.3 m	45 kg	Mammals	1

Table 2: Structure data using decision forest

## 3 Malware Distribution on Different OS

### 3.1 Malware Attack on Different OS

#### 3.1.1 Windows

Figure- 20 demonstrates the distribution of malware assaults across different operating systems. The most extensively used computer operating system in the world is Microsoft’s Windows and because of its wide usage, it is also prone to various malware invasions. In 2019, 114 million new malicious programs were created, according to AV Test’s 2019/2020 Security Report, with Windows systems accounting for 78.64 percent of all attacks. The COVID-19 epidemic has been the cause of the rise of millions of new malicious programs. Many hackers have taken advantage of the uncertainty and instability to propagate malware. As a result, the percentage of malware targeting Windows computers grew to 83.45 percent in the first quarter of 2020.

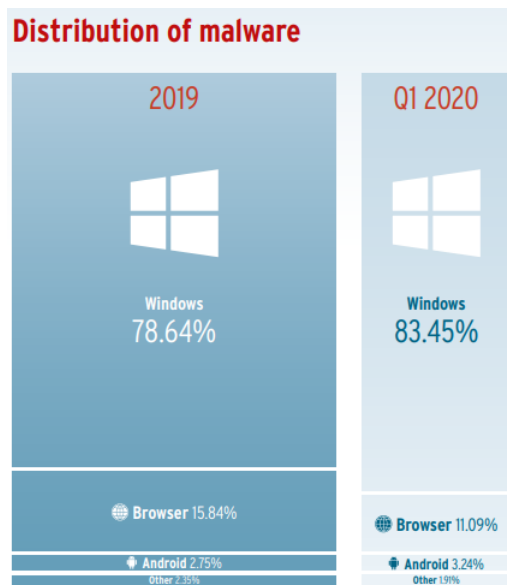


Figure 20: Distribution of malware attacks in different OS

As previously stated, Microsoft systems are the primary target of malware attacks because the profit margin is much higher due to the large user base, and thus

cybercriminals have been following the laws of economics.

**Distribution of malware under Windows in 2019**

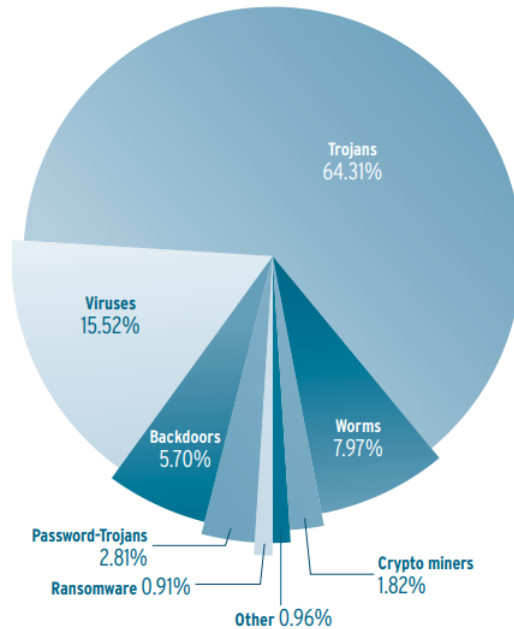


Figure 21: Distribution of malware under windows

Malware distribution under Windows is seen in Figure- 21.

### 3.2 Why Linux is Secured

Linux has a small user base compared to other operating systems but has a better track record compared to windows when it comes to security. For increased reliability, more business corporations are migrating to Linux. We have attempted to investigate some of the reasons why Linux is more secure.

**File Permissions Handling:** Running an infected file is one of the most common issues that users experience. It is simple to run a file that has a negative impact on a computer. Because this is not a separate and independent process, Linux normally does not process executable files without specific permission from the legitimate user. Linux distinguishes between root and non-root users, preventing excessive file handling risks.



Virus Removing Mechanism: Because many Linux distributions split the root user and the usual user, if a problem arises with one of the users, it is simple to remove that account and create a new one while the root user is largely unaffected. Linux also comes with an anti-malware platform that protects against viruses, trojans, rootkits, and other malware.

Open-source: The Linux kernel is free and open-source software, and as such, the code is created and maintained by a group of dedicated individuals known as the "community." The quality of the code is assured when more people see it and review or test it. Despite the fact that this notion applies to all open-source software development, it is strictly adhered to on Linux. Developers will continue to provide remedies for security vulnerabilities/flaws in the form of "patches" in addition to entry-level quality control. Figure- 22 demonstrates the Linux kernel development cycle.

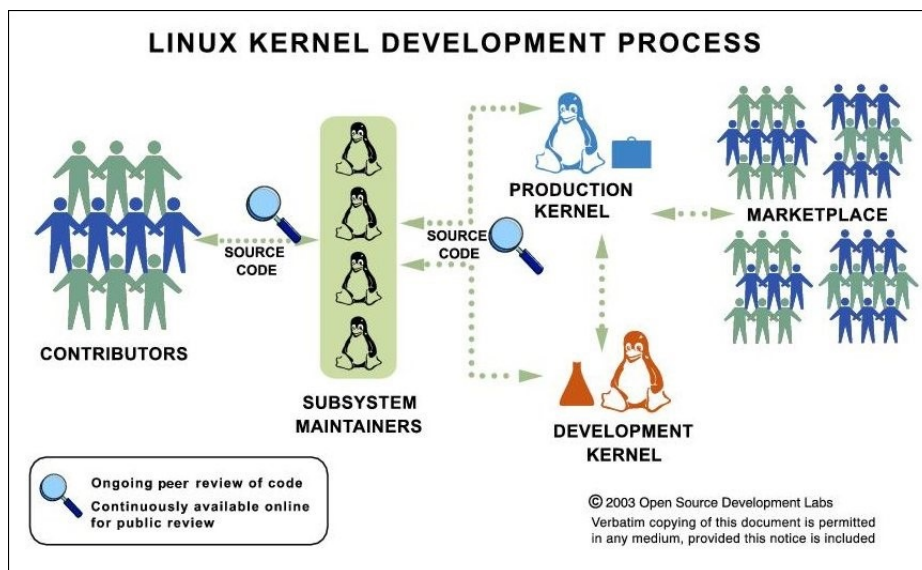


Figure 22: Linux kernel development

Memory management: User and kernel space are well separated in Linux. Users will not see the real physical address assigned to any process here. Try printing the addresses of the parent and child processes generated with the fork() system function. Both the parent and child processes will have the same virtual address.

The reason is that any application can access only the virtual address that is mapped to the physical address. As a result, no harm can be done to the real physical address space. This method of memory management makes Linux more secure. The reason is that any application can access only the virtual address that is mapped to the physical address. When a user application makes a system call, a “software interrupt” from user space to kernel space is triggered, causing a delay. When converting a virtual address to a physical address, the translation procedure will cause some delays as well.

Updated System and Software: The system can also be vulnerable to the old version of any software. It is mandatory to update the system as well as software to keep up to date. As bug fixes and updates can fix serious security issues, it is always recommended that the whole system be updated frequently for better performance along with security measurement. Linux, on the other hand, makes it simple to receive updates and security fixes for both the system and the applications with only a few commands. It’s all because of the package managers.

### **3.3 Why Android is Vulnerable**

Windows may be the most widely attacked Operating system but in terms of security gaps, Android is the most vulnerable one. According to the AV-TEST 2019 security report, Android was the most insecure operating system. Any smartphone is vulnerable to security flaws, but Android phones are far more likely to be hacked. Unlike Apple’s iOS, the Android operating system is open-source, which means anybody can make changes to it. If a developer makes a mistake, a hacker will have an easier time locating and exploiting security flaws. In comparison to Apple, Android has a bigger market share. This makes it easier and more profitable for hackers to write and distribute Android malware in order to collect sensitive information from users.

People in the present era are always linked to the internet. As the use of technology becomes more widespread, security concerns are becoming more prevalent.

Malware threats have been estimated to have afflicted Android smartphones more than any other. The results of that study are summarized in figure- 23.

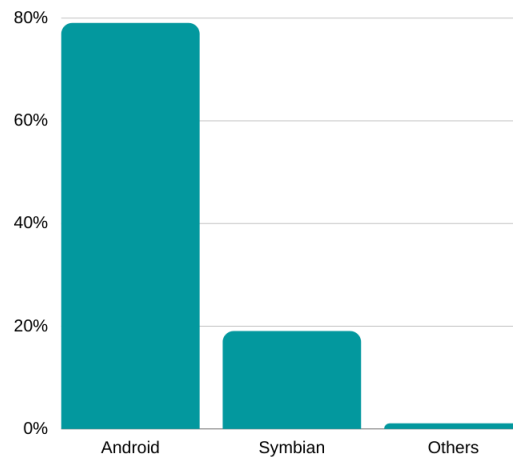


Figure 23: Most vulnerable operating system

It is said that a total of around eighty percent of malware threats affect Android OS. Where the percentage for Symbian is just below 20 and the remaining affects other operating systems.

It is vital to analyze the basic structures of Android, specifically how it is created and the workflow, in order to point out its flaws. The android architecture and working procedure of the operating system are explained.

As the overview of the architecture is seen above in figure- 24, there is another study that provides information about vulnerabilities in different layers of android architecture.

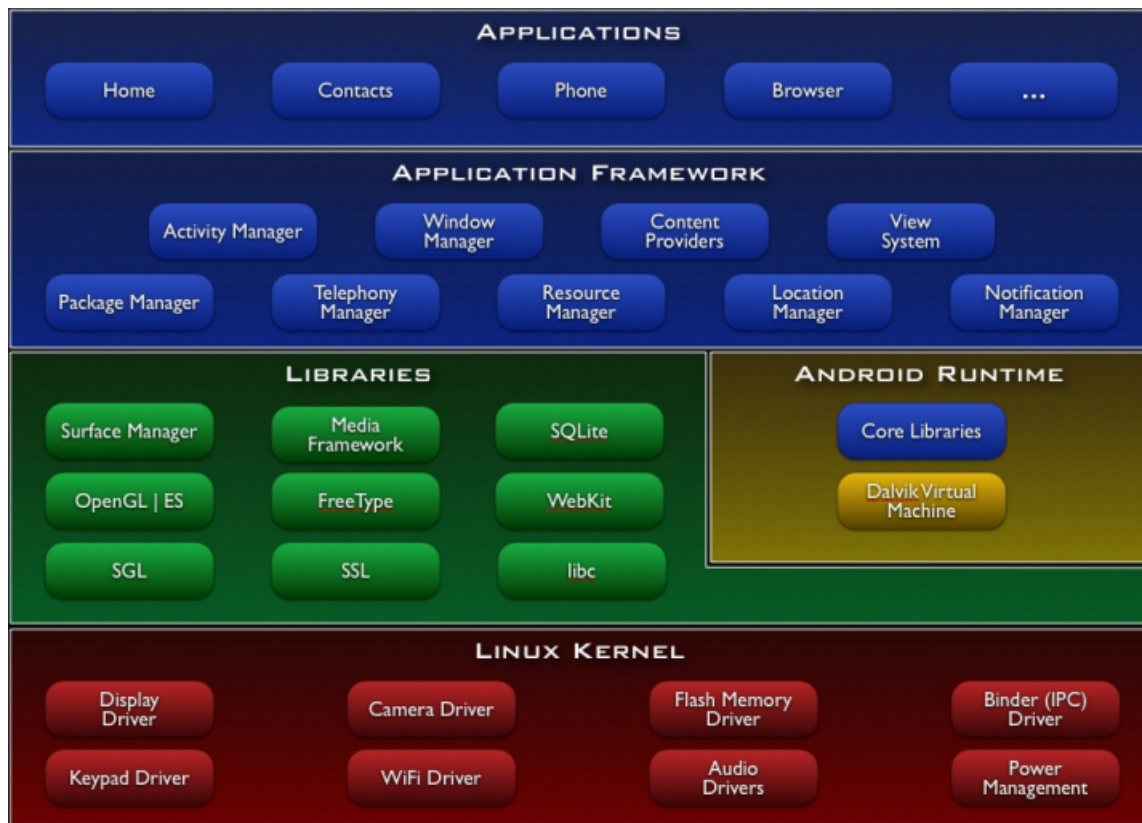


Figure 24: Android architecture

It is found that the application framework level of android architecture is the most vulnerable layer and the Linux kernel layer is the least vulnerable layer. The vulnerability comparison among different layers can be found in figure- 25

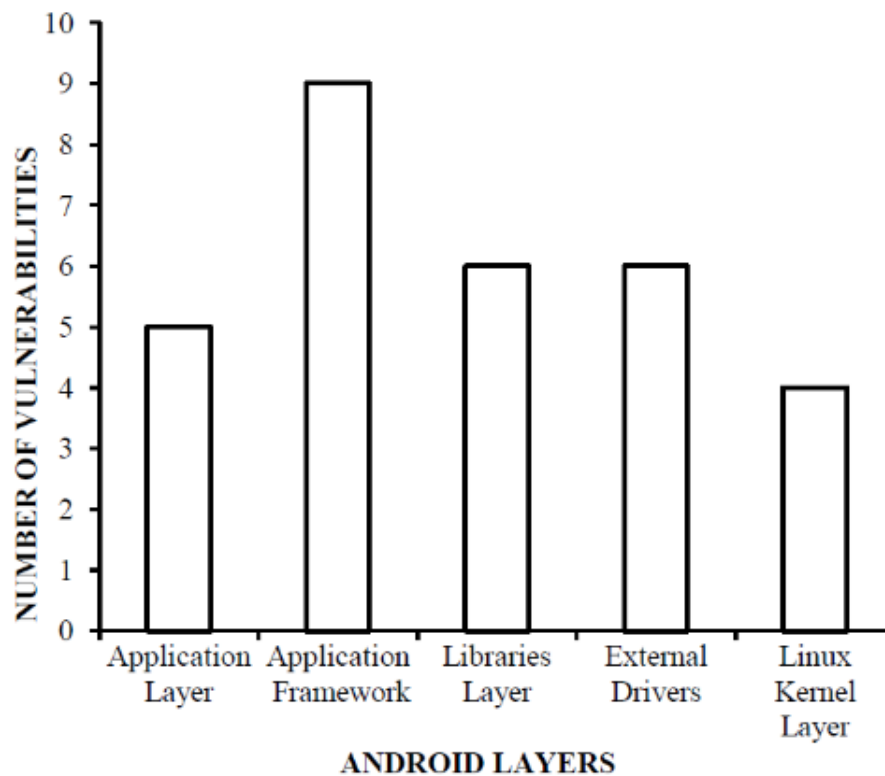


Figure 25: Android vulnerable in each layer

## 4 Malware Analysis

### 4.1 Malware Analysis Techniques

Malware analysis might be static, dynamic, or hybrid: a combination of the two.

#### 4.1.1 Static Analysis

In this analysis, we do not observe the internal structure or get access to the system. Instead, it looks for the behavior and characteristics of any malicious content in a file. This process is fast and usually cost-friendly as well. Static analysis is mainly used to identify if any malware exists in the system or not. The static analysis uses some technical signs like file names, hashes of data, domains, and file header data to identify malware. There are also some disassembler tools and network analyzers that can effectively check malware without executing the

code. These tools can provide greater knowledge of how malware could operate [20].

However, this analysis may not always turn up to be optimal. Some of the attacks may go unnoticed due to the lack of information. A basic static analysis cannot systemically identify a malicious file if it can create a string and subsequently downloads based on the string [21].

#### 4.1.2 Dynamic Analysis

Dynamic analysis can provide a wider knowledge, in most cases full knowledge of the file’s behavior. It works in a ‘sandbox environment’ where it analysis any suspicious behavior inside a code. This allows security experts to observe the malicious file without altering the system or interrupting it [22].

Experts can find many benefits from dynamic analysis, especially because it can see the malware’s nature and type. Also, it saves time by eliminating reverse engineering a file in order to discover harmful code.

However, dynamic programming has many limitations. The attackers are clever enough to know about the techniques that the sandbox environment may use. Usually, the codes are written in a way that can bypass the detection criteria and remains inactive. Then it executes only when the code is running [21].

Following (Table- 3) are the differences between static and dynamic analysis [32].

Static Analysis	Dynamic Analysis
1.Comparatively faster and safer	1.Time-consuming and vulnerable
2.Easy analyze multi-path malware	2.Difficult analyze the multi-path malware
3.Ineffectual for advanced malware	3. Efficient against all malware
4. Low level of false-positive	4. High level of false-positive
5. Accuracy is high	5. Accuracy is low

Table 3: Difference between static and dynamic analysis

### 4.1.3 Hybrid Analysis

To deliver the best of both approaches, hybrid analysis combines fundamental and dynamic techniques [22]. A simple static analysis fails to detect sophisticated malware and can bypass the sandbox technologies easily. The hybrid analysis provides better security by combining both static and dynamic techniques. Hybrid analysis can be used to discover malicious code that may try to hide as well as check the software where they had gone undetected before. Hybrid analysis helps in the detection of unknown threats, including those originating from the most sophisticated malware [23].

## 4.2 Malware Detection

Malware detection techniques are broadly classified into three types: signature-based, heuristic-based, and specification-based [26]. To protect the system from malware, these techniques both detect the system and take action against the malicious files.

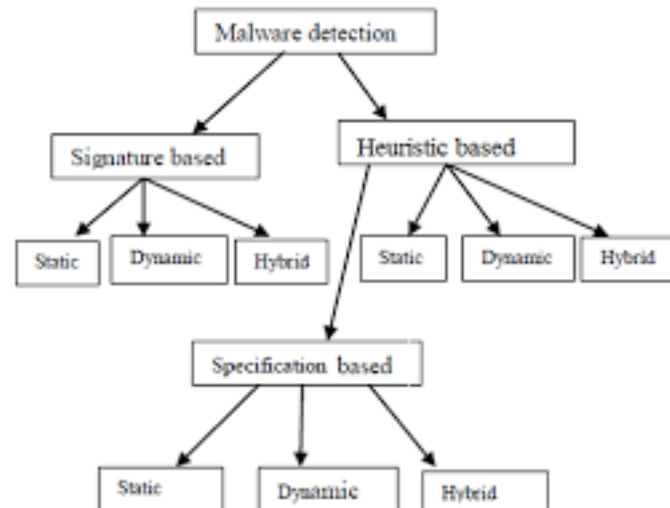


Figure 26: Malware detection techniques

### 4.2.1 Signature-based detection

Signature is a sequence of bits that is produced in the code when malware is being created. It can be used afterward to identify the class or type of malware. This detection technique is widely used in anti-virus products. The antivirus program disassembles the code of the infected file and looks for malware family-related patterns. Signatures generated by the malware are usually stored in a database and then compare it with the file to check whether malware exists. This detection technique is also known as ‘string or pattern scanning’ [24]. Static-based detection can be both static, dynamic, or hybrid.



	Advantages	Disadvantages
Signature-based detection	Easier to execute Can identify fast Can access broadly Find comprehensive malware information	Not being able to identify polymorphic malware Replicating data in a massive database
Heuristic-based detection	Detecting previously unknown sorts of malware assaults Detector of data flow dependencies Polymorphic malware detection	For behavioral patterns, storage complexity is important. Time complexity
Specification-based detection	Both known and undiscovered malware, as well as novel malware, can be identified. False-positive rate is low	ineffective in detecting fresh malware creating specifications takes a long time False positive rate is high

Table 4: Advantages and disadvantages of various detection techniques

### 4.2.2 Heuristic-based detection

Heuristic-based detection detects and differentiates unusual system activities. This allows the system to discover the unfamiliar circumstances which is very effective in detecting both known and unknown malware. The heuristic-based detection approach involves two phases. In the first phase, the system’s behavior is examined in the absence of an attack, and a log of vital information is kept that will be confirmed and checked in the event of an attack. In the second phase, the differences are observed to determine the malware’s family and classification [24].

### 4.2.3 Specification-based detection

In a specification-based detection technique, applications are monitored and checked for usual and aberrant behavior based on their specifications. This technique is similar to heuristic-based detection techniques, but the primary distinction is that heuristic-based detection techniques use machine learning and AI methods to detect the valid and invalid activity of a legitimate program, whereas specification-based detection techniques are based on an analysis of the behavior described in the system specification [25]. This method entails a manual comparison of a system’s regular operations. Reducing false positives and raising false negatives, it overcomes the limitations of heuristic-based approaches.

#### **4.2.4 Machine learning-based detection**

Each machine learning software has a big influence, especially when it comes to the categorization of Android malware, which is still considered a new study topic [29]. A machine learning-based malware detector can assess whether or not an app is harmful [28]. The majority of the machine learning papers evaluated focused on classification, with only a handful focusing on clustering. Those studies that focused on categorization assess if applications are malicious or benign, and they continually monitor distinct patterns and attributes that might indicate a device's state, such as its battery level and memory usage. An app is first launched in these tests before machine learning techniques are used to identify whether the app is malicious or benign. Research that has concentrated on clustering, on the other hand, classified distinct forms of malware into families or groups based on behavioral or characteristic similarities [27].

#### **4.2.5 Cloud-based detection**

Cloud-based detection appears to be the way of the future for mobile security that is quick, efficient, and effective [30]. Having a smart system that only analyzes malware statically and dynamically will prove to be a formidable opponent for malware creators.

## 4.3 Malware Prevention

Policy, awareness, vulnerability mitigation, and threat mitigation are the four primary elements of prevention [31].

### 4.3.1 Policy

Ensuring policies address malware prevention as a foundation for adopting preventative controls. Malware prevention policies should be flexible in policy execution and decrease the need for frequent policy revisions, while simultaneously being specific enough to clearly define the policy's goal and scope. Some of the common malware policies are:

1. Before external media can be utilized, it must be scanned for infection.
2. E-mail attachments, especially compressed files (such as .zip files), should be saved to local disks or media and checked before being opened.
3. Users should be restricted from using admin privileges, which helps to reduce the privileges accessible to malware that is introduced to systems by users.
4. Systems should be kept up-to-date with OS and application upgrades and patches.
5. Requiring approval of firewall configuration modifications through a formal process.

### 4.3.2 Awareness

To reduce the frequency of incidents caused by human error, it is necessary to establish and maintain general malware awareness programs for all users, as well as particular awareness training for IT professionals directly involved in malware prevention-related tasks [31].

1. Suspicious e-mails or e-mail attachments from unknown or known senders should not be opened.
2. Popup windows in web browsers should not be clicked.
4. Web sites that are at least somewhat likely to contain malicious content should

be avoided.

5. It is not recommended to open files with file extensions that are known to be affiliated with malware.
6. Additional security control mechanisms should not be disabled.

### **4.3.3 Patch Management**

Patch management is the technique of proactively hardening software against specific security flaws before they may be exploited by hackers. Tracking and overseeing software patch releases is part of this procedure. The following are the stages of patch management:

1. Detecting and identifying a bug
2. Creating a bug-fixing code solution
3. In a sandbox, testing the patch.
4. Approving the patch
5. Documenting the patch code
6. Releasing the patch to end-users
7. Monitoring the patch release

Patch management is critical since it protects your application from cyber-attacks. When delivering a new patch, developers must exercise caution because it may disrupt the device's other programs and functionality. Patches also protect against software performance concerns and misalignment of platform versions [36].

### **4.3.4 Least Privilege**

Least Privilege is the notion of constraining user and application access to privileged accounts via various restrictions and technologies without affecting productivity or necessitating IT support. Only the bare minimum of rights should be granted to a user who asks for access to a resource, and they should be granted for the shortest time possible. The reason behind this strict limitation is that letting

users gain access more than absolutely necessary would enable them to modify data in unwanted ways [35].

#### **4.3.5 Other host hardening measures**

Deploying additional host hardening measures can help lower the risk of malware attacks.

1. Disabling network services that are no longer in use and may contain vulnerabilities.
2. Getting rid of file shares that are not secured.
3. Default usernames and passwords for operating systems and applications need to be removed or changed.
4. Disabling binaries and scripts from auto-execution.

#### **4.3.6 Threat mitigation**

Many tools, platforms, and techniques exist to mitigate and limit cyber dangers, easing part of the strain and empowering businesses to defend themselves against hackers. Threat mitigation employs a wide spectrum of expertise, but systems administrators are frequently the ones who put threat management methods into action. Patch management, intrusion detection, and post-attack cleanup are all tasks that sysadmins often handle.

#### **4.3.7 Antivirus Software**

The most prevalent tool for dealing with malware threats is the antivirus or anti-malware software. These types of software are widely used for Identifying common types of malware, disinfecting files, scanning critical system components, and Monitoring real-time system activity to look for suspicious behavior.

### **4.3.8 Spyware detection and removal utilities**

Spyware is easier to prevent but difficult to detect. Detection of spyware, therefore, should get higher priority.

1. Monitoring the activity of the apps that are most likely to be used to install spyware on computers, such as Web browsers and email clients.
2. Scanning files, RAM, and configuration files for recognized spyware on a regular basis.
3. Preventing spyware installation via a variety of methods such as pop-up ads, tracking cookies, browser plug-in installations, and browser hijacking.

### **4.3.9 Intrusion prevention systems**

IPS stands for intrusion prevention system, and it is a network security solution that identifies threats and illegal access. It notifies your security staff in order to assist them in preventing assaults. An IPS can also initiate actions such as access point closures and firewall configuration changes. An intrusion prevention system (IPS) is a hybrid of an intrusion detection system (IDS) and a firewall. IPS and IDS are sometimes included in a network security solution that resembles a firewall. The arriving network packets are compared to a database of cyberattack patterns. Both systems flag the packet if they detect a match [33].

### **4.3.10 TPM and Secure Boot**

Secure Boot is an essential security feature deployed with the intention to prevent malicious software from loading when a computer boots up. Computers that came out in recent years are capable of secure boot. There could be cases in which a computer is unable to use secure boot because of settings and these settings can be modified in the firmware or BIOS of that particular computer.

There are two types of security: software and hardware. When implemented right, software security is an effective way to keep malware out of a system. Since software by nature is malleable there is always a possibility that it will be exploited

to get access to sensitive data. Hardware security is hardcoded, as the term implies and it is not possible to alter the cryptographic keys. One of the ways to ensure effective hardware security is through TPM.

TPM is short for Trusted Platform Module and its purpose is to safeguard data necessary to authenticate the computer. By securely producing and preserving cryptographic keys, a TPM increases the security of the computers. People were not that familiar with TPM until Windows 11 came into the picture with a bunch of requirements and one of them happens to be a TPM 2.0 chip in the computers. The idea of a trusted platform module is not new though and it goes way back. There were computers in 2005 that had TPM chips. Because of dire security threats, it has been made mandatory for computers that want to get upgraded to Windows 11.

TPM can save any component of the secret needed for decryption, including passwords, certificates, and encryption keys. This information is also stored in real hardware rather than software by the TPM. This means that a software assault will not be able to divulge the TPM's secrets. Through secure boot and TPM 2.0 chip requirements, the OS will eradicate an entire set of malware attacks that aim to take over the computers by getting into the system before it boots up.

#### **4.3.11 Firewall and routers**

A router is a device that carries data across networks, whereas a firewall is a device that filters data before it is delivered over a network. Routers are frequently connected to at least two networks; a firewall prevents incoming requests from obtaining private network resources by running on a separate computer from the network [34].

In short, Organizations should be mindful, however, that no matter how much effort they put into preventing malware events, incidents will still happen (e.g., previously unknown types of threats, human error).

## 5 Related Works

Many research attempts have been made to prevent various malware threats, using multiple ways to detect malware presence. Numerous studies have sought to provide a summary based on the information gathered. This portion of the paper will showcase papers that reflect a broad view of malware-related research.

Herron et al. [61] studied and analyzed four machine learning algorithms namely k-means, random forest algorithm, support vector machine algorithm, gaussian naïve bayes algorithm, and algorithm, where the features are selected from manifest file permissions of Android for classification of malware and benign. A total of 5,243 samples were studied and each algorithm was showing accuracy, recall, and precision rates over 80 percent margin, concluding that Random Forest showed the best performance with an 82.5 percent of precision rate and 81.5 percent accuracy rate.

The study conducted by Hahn et al. [62] worked on adding attribute sets on Android. A total number of 11 attributes were gathered on 10,000 apps. They set up a comparative analysis that provides a single attribute sets ranking based on the performance of detection.

Anderson and Roth [63] provide a dataset that can detect malicious files intended for Windows Operating System. The collection contains information retrieved from binary files of 1.1 million, including training examples of 900,000 (within which 300,000 are malicious, 300,000 are benign, and 300,000 unlabeled) and test samples of 200,000 (within which 100,000 are malicious, 100,000 are benign).

Sewak et al. [64] explored a malware detection system based on Deep Learning. The researchers improved the previously mentioned work by 99.21 percent accuracy (previously 98 percent) and with a 0.19 percent False Positive Rate (previously 1.07 percent).

Joyce et al [65] created a dataset named MOTIF also known as malware open-source threat intelligence Family which consists of a total number of 3,095 malware



from a number of 454 families. The researchers claim their work to have the “most diverse public malware dataset”. The dataset’s antivirus majority voting accuracy is only 62.10 percent, while their comparative tool’s accuracy is only 46.78 percent.

SIGPID, the detection of malware based on permission usage was developed by Li et al. [66]. The authors developed pruning of three-level by the permission data mining. This technique helps specify the significant permissions. A total of 22 permissions were found as significant. SVM algorithm was used and all the confusion matrix elements were 90 percent. The software was successful in noticing 93.62 percent of the trained data set and 91.4 percent of the further malware representatives.

Mahindru and Singh [67] studied 11,000 Android applications and obtained a total of 123 dynamic permissions. From these extracted data they evaluated numerous machine learning algorithms namely random forest, k-star, decision tree, simple logistic, and naive bayes.

Arslan et al. [68] looked at identifying extract requested permissions. They claimed that the accuracy of their model is 91.95 percent.

Milosevic et al. [69] presented a couple of machine learning approaches intended for mobile app static analysis. The foremost way is called source code-based classification. It performed 95.1 percent on the f-score. Another approach is to utilize permission names. This only performed 89 percent of the f-score.

## 6 Proposed Approach

This section briefly discusses the work done, the methodology used, different environmental setups, data collection procedures, design, and testing methods.

### 6.1 Methodology

To conduct malware analysis, two directions were followed in this paper. Firstly, human factor analysis in malware, where the participation of general people is measured in being a victim of malware attack. Secondly, study open-source techniques and perceive a suitable one by analyzing the findings in different mediums. An experiment was done among general real-life users. By following an open-source code a technique was implemented. This method generated a malicious link, which provides the project builder (the person who executes that program at that time) access to different features in a targeted user's machine. When the user clicks on the link, they are asked for the camera, microphone, and location access of the system. After accepting the executors got various information from those access points. This experiment could be done by sending an APK file (for android use) or an execution file (for desktop use). In this work, this method is chosen to send links because of the simplicity for users and researchers. These malicious links were sent to relatives and friends to test whether they can be manipulated to click the link. Some people fell into the trap and clicked. After getting the information, they were told about the work and what they have done by giving that permission information. This experiment was held to show that people can be manipulated and fall into such traps. In this experiment, no information was stored any further after sending it back to the victim. Finally, everyone who clicked the link was provided a form to fill up and gave useful information. In the next part, open-source datasets, as well as open-source tools and techniques, were used to analyze malware from different aspects. This work also helped validate different open-source techniques developed with older technology with the current system. The main focus of this work was to look at existing malware detection and possible

prevention techniques. A lot of currently existing tools are studied that are used to detect malware efficiently. Another technique was used to extract permission from applications and create a dataset that is fed into different machine learning algorithms. Finally, these results are analyzed from various angles.

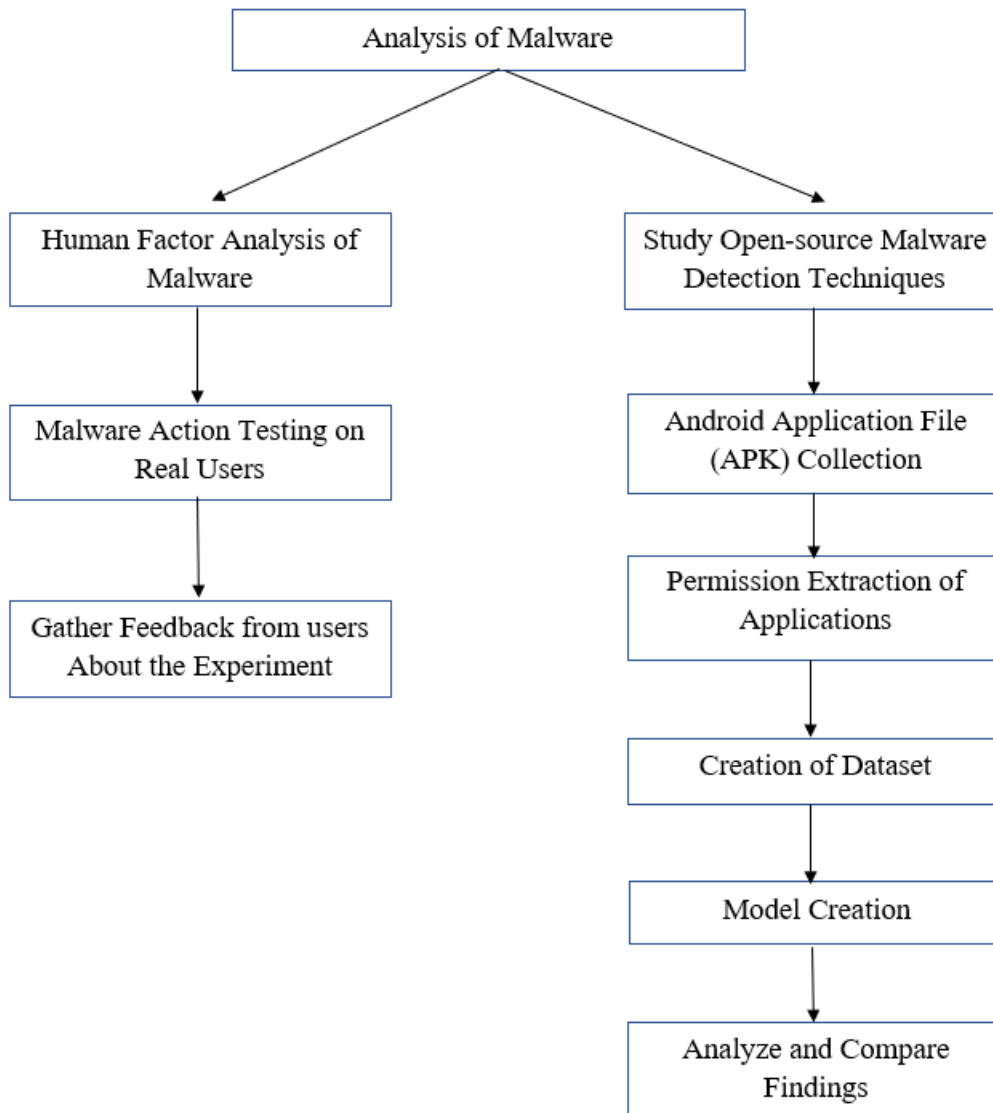


Figure 27: Workflow of the Paper

## 6.2 Environment

The main experiments (executing the detection techniques and feature extraction) were held in the Ubuntu operating system. Genymotion Desktop was used for an-

droid emulation. It is an Android emulator that provides a comprehensive set of sensors and functions for interacting with a virtual Android environment. This software allows testing the Android apps on a variety of virtual devices for development, testing, and presentation. While executing and testing many packages and libraries were adapted to the current system of testing to accomplish desired results. Furthermore, machine learning-based algorithms are used in Google Colaboratory. This platform gives the opportunity to handle big datasets and analyze the outputs more.

### **6.3 Data Collection**

For android malware analysis, Kharon Malware Dataset [39], CICInvesAndMal2019 [38], MalwareBazaar [40], and various open-source malware present on Github and other platforms are used. People created most of these samples with bad intentions as well as some are created for helping researchers to conduct quality works. Therefore, these samples gave us a good initiative to analyze the latest malware with previously developed tools and techniques. From the above-mentioned sources, android executable files (APK) are collected. These applications are used to create the dataset. The PACE Project [48][49] was used to extract the features from the original data. This dataset is fed into various machine-learning algorithms. A total of 1200 applications are collected. The dataset is created using 1168 apk files (Among these applications, 602 are malware and 566 are benign applications) consisting of extracted features and different permissions of those apps. Each application has 948 features. Every property contains a value of either 0 or 1, where 1 means that the application uses that specific permission and 0 means the opposite.

### **6.4 Design and Testing**

It is mentioned that open-source tools and techniques with various data are used totally in the experiments. To run a particular tool, proper guidelines need to be followed in order to execute and analyze. Many tools are designed with an older

system that is currently out of order, and those needed to be validated and run in the latest system. Various packages are unsupported; thus, the equivalent of those are used and got started. Otherwise, most of the tools that are used were as it was. Before starting the analysis of malware detection, an open-source program was implemented that provided a shareable link. This link was given to different users in order to see a wider scenario of malware accessibility. If anyone clicks the link, information (specifically camera, microphone, location access) is bypassed illegally according to the needs. This software was run in 2 different Debian distributions (Kali and Ubuntu) to see that these types of programs are accessible. After clicking the link, the users were given a form about what they faced and their opinion about the consequences of such attacks in real-life situations. In the next part of the analysis, a customized ransomware application is created following SARA [37]. When the application file (APK) is installed on the android devices, the system prompts a window asking for a password to unlock and use the device. In a real-life scenario, one can only gain this password by sending money to the application developer. This application is used just for educational purposes. There was no harm intended while experimenting with this application. Quark-engine [41] to check the APK file in various aspects. Then a comparison of that app with recent ransomware was done. This comparison is done to demonstrate the smallness and simplicity of the customized application that is used to test different techniques of malware analysis. Droidbot [42][43], Androwarn [44], APKStat [45], APKiD [46], and DroidLysis [47] are some of the examples that are used to analyze the edited application. There were three virtual android devices with various types of software installed. With Check all APK's [58] and Drozer [56] [57], these devices and their packages were studied totally and generated a report. The Kharon Malware Dataset [39] and CICInvesAndMal2019 [38] dataset, MalwareBazaar [40], and various open-source malware (present on Github and other platforms) were used with the PACE Project [48][49], Drebin [50][51] and CSBD [52] [53] [54] [55]. The collected apk files are used in the PACE Project [48][49] to extract the features and put them into a different dataset as a comma-separated

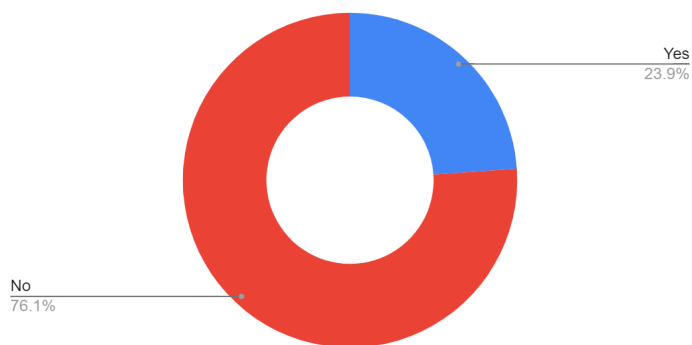
value list. This dataset is used to analyze malware from machine learning-based algorithms. Different approaches like Naive-Bayes Algorithms, K-nearest neighbors, Decision Tree Algorithm, Random Forest Algorithm, TensorFlow Decision Forests Algorithm were the main focuses of this research work. Confusion matrix, accuracy, differences of various permission analyses between benign and malicious software, the importance of features, and many more inspections are done and studied.

## 7 Result Analysis

In the first part of the conducted experiment, a demonstration is tried to make that human can be manipulated to be a victim of malware attack. If a person knows the sender then he/she is most likely to believe what the sender has sent him/her. In the work, a total of around 200 people (who are known and communicated with on a daily basis) were sent the malicious link. Out of the total number, only 46 people clicked the link when they are offered. A form was forwarded to each of them to get their feedback.

a. Assuring the Information of the Provided Link

Count of Did you ask the sender about any details of the link before clicking



b. Closeness and Manipulative Rating Scenario (Rating vs Number of Response)

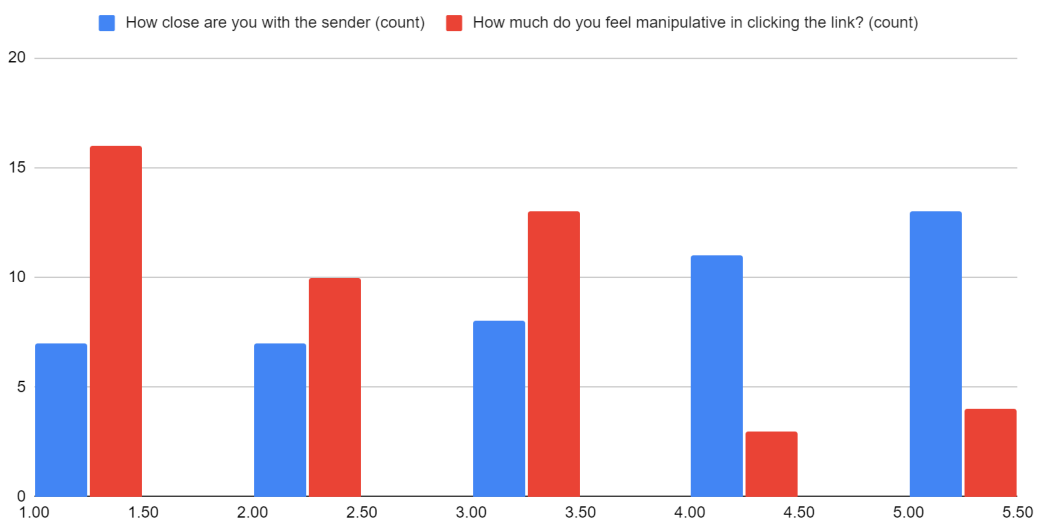


Figure 28: Questionnaire from the Experiment



Figure- 28 shows that people who know each other do not even bother asking what kind of information they are going to get after clicking the link. More than one-third of the respondents replied that they did not feel the necessity of asking for the information beforehand. Figure 28(b) represents the manipulation of being the victim with the relationship status of the sender. The people who gave higher ratings are more likely to believe that the sender is not manipulating them. Whereas the lower relationship rating indicates that people think they are being manipulated. This human factor experiment can be done in more detail with various other sets of questions and their feedback.

The customized app (by following SARA [37]) is installed into the virtual android devices that are set up for the experiments. All of the devices show similar results of locking out of the devices as pictured in Figure- 29.

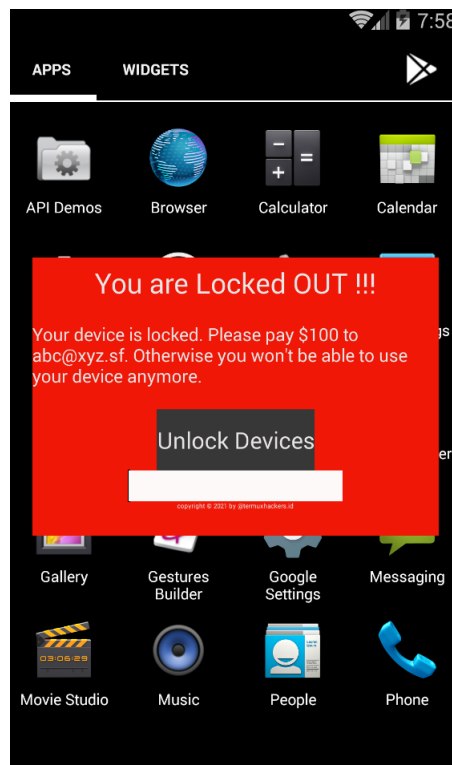


Figure 29: A simple ransomware example

This behavior locks out the user from the device and prompts a password window. The users cannot use that device any further without the password. This application's behavior will be analyzed further.

The edited application is checked thoroughly by Quark-engine [41]. The label-based report is shown in Figure- 30.

[-] Total Label Found: 34  
 [-] Rules with Label which max confidence >= 80%: 1

Label	Description	Number of rules	MAX Confidence %	AVG Confidence	Std Deviation	# of Rules with Confidence >= 80%
collection	-	76	20	19.74	2.28	0
command	-	27	40	22.22	6.29	0
sms	Read/Write/Send sms content	26	20	19.23	3.85	0
network	-	24	40	20.83	4.0	0
file	-	23	20	20.0	0.0	0
reflection	-	20	60	24.0	10.2	0
telephony	-	16	20	20.0	0.0	0
wifi	-	13	20	20.0	0.0	0
location	Leakage of Location of the device	11	20	18.18	5.75	0
control	-	10	100	30.0	24.08	1
record	-	8	20	20.0	0.0	0
accessibility service	Use Accessibility Service to perform user actions	7	20	20.0	0.0	0
callog	Retrieve or manipulate sensitive data from call log	4	20	20.0	0.0	0
calendar	Get calendar information as calendar event	4	20	20.0	0.0	0
socket	-	3	20	20.0	0.0	0
privacy	-	3	20	20.0	0.0	0
http	Use http to send sensitive data	3	20	20.0	0.0	0
camera	-	3	20	20.0	0.0	0
so	Load native libraries(.so)	2	20	20.0	0.0	0
phone	-	2	20	20.0	0.0	0
power manager	-	1	20	20.0	0.0	0
pin	-	1	20	20.0	0.0	0
permission	-	1	20	20.0	0.0	0
packer	-	1	20	20.0	0.0	0
notification	-	1	20	20.0	0.0	0
lock	-	1	20	20.0	0.0	0
exec	-	1	40	40.0	0.0	0
evasion	-	1	40	40.0	0.0	0
dexClassLoader	-	1	20	20.0	0.0	0
connection	-	1	20	20.0	0.0	0
applications	-	1	20	20.0	0.0	0
admin	-	1	0	0.0	0.0	0
accounts	-	1	20	20.0	0.0	0
account	-	1	20	20.0	0.0	0

Figure 30: Confidence Measurements of Label

These labels are indicators of various topics of the malware’s aggressiveness. As this is a simple application that was tested, the values are less significant as the number of rules is quite smaller.

Then this application is compared with a similar application (ransomware) present in the Kharon dataset [39]. The behaviors of both applications can be studied based on the max confidence of rule labels. This is shown in Figure- 31.

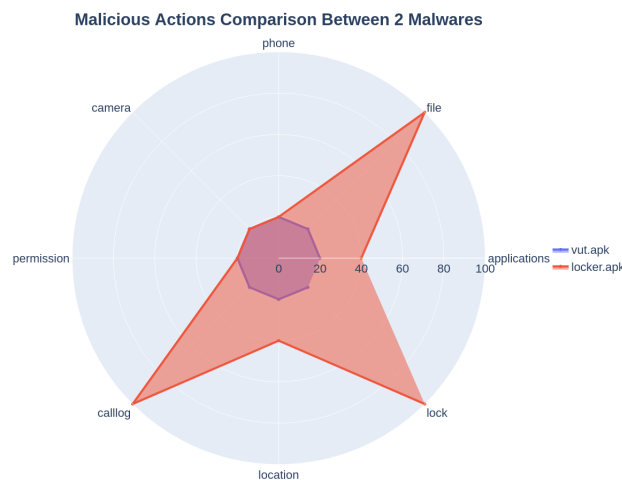


Figure 31: Comparison of Malicious Actions Between Two Malwares

It is clear in Figure- 31 that the confidence level is very low for our application. Thus, the comparison with live malware is clearly visible in the chart. This also pictures the wide range of malware activities in the current context of the world. There are a lot of features that can be analyzed via this technique. The performance index of malware can be further documented in order to study and extract the behavior of malware.

The Androwarn [44] tool gives a lot of opportunities to explore application details. After testing the edited app, application information, analysis results, apk files, and XML information with APIs used are found. Figure- 32 shows the permission analysis of the app.

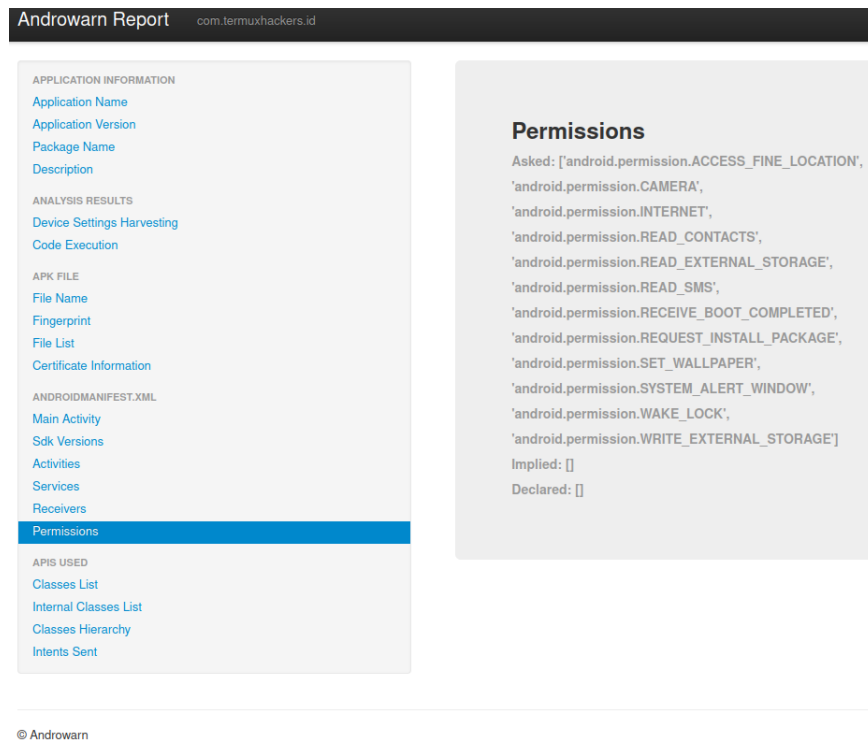


Figure 32: Permission Analysis from the Detailed Report

This tool can be used in various other information gatherings, but the purpose of this work is to look for permission extraction. Here a total of 12 permissions are found that are used by the app.

The APKStat [45] tool also gives a summary report gathered from the apk files. It analyzes the permission request information, activities name, services name, receivers name, and providers name with a summarized report. Figure- 33 shows the report outlook.

```
Permissions Requested:
android.permission.SYSTEM_ALERT_WINDOW
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.SET_WALLPAPER
android.permission.READ_EXTERNAL_STORAGE
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.READ_CONTACTS
android.permission.READ_SMS
android.permission.ACCESS_FINE_LOCATION
android.permission.WAKE_LOCK
android.permission.INTERNET
android.permission.REQUEST_INSTALL_PACKAGE
android.permission.CAMERA

Names Of Activities:
com.termuxhackers.id.MainActivity

Names Of Services:
com.termuxhackers.id.MyService

Names Of Receivers:
com.termuxhackers.id.BootReceiver

Names Of Providers:

Names Of Activity Aliases:

Number Of Permissions Requested: 12
Number Of Activities: 1
Number Of Receivers: 1
Number Of Providers: 0
Number Of Services: 1
Number Of Activity Aliases: 0
Number Of Possible Domains: 0
Number Of Possible IPs: 0
```

Figure 33: Summary report of the Application

This report shows the permissions used by the application. Here also it is seen that 12 permissions are requested. Here is a similarity found between the APKStat [45] and the Androwarn [44] tool. They are both capable of extracting 12 permissions of the same application.

APKiD [46] provides information about how an APK was made. It identifies numerous compilers, mystifiers, combiners, and many more attributes contributing to the making of an application. Here the customized application is tested with Kharon datasets [39] apps to get an idea of what the information will look like. Figure- 34 demonstrates that information.

```
[+] APKiD 2.1.2 :: from RedNaga :: rehnaga.io
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/31801dfbd7db343b1f7de70737bdbab2c5c66463ceb84ed7eeab8872e9629199.apk!classes.dex
|-> anti_vn : Build.FINGERPRINT check, Build.MANUFACTURER check, network operator name check
|-> compiler : dx (possible dexmerge)
|-> manipulator : dexmerge
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/2ee72413370c543347a0847d71882373c1a78a1561ac4faa39a73e4215bb2c3b.apk!classes.dex
|-> anti_vn : Build.BOARD check
|-> compiler : dexlib 2.x
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/8a918c3aa53ccd89aaa102a235def5dcffa047e75097c1ded2dd2363bae7cf97.apk!classes.dex
|-> anti_vn : Build.MANUFACTURER check, Build.TAGS check
|-> compiler : dx (possible dexmerge)
|-> manipulator : dexmerge
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/vut.apk!classes.dex
|-> compiler : dexlib 2.x
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/f75678b7e7fa2ed0f0d2999800f2a6a66c717ef76b33a7432f1ca3435b4831e0.apk!classes.dex
|-> compiler : dx (possible dexmerge)
|-> manipulator : dexmerge
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/54f3c7f4a79184886e8a85a743f31743a0218ae9cc2be2a5e72c6ede33a4e66e.apk!assets/legacy!classes.dex
|-> compiler : dx
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/54f3c7f4a79184886e8a85a743f31743a0218ae9cc2be2a5e72c6ede33a4e66e.apk!classes.dex
|-> compiler : dexlib 2.x
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/919a015245f045a8da7652cefacc26e71808b22635c6f3217fd1f0deb6d1d4330.apk!classes.dex
|-> anti_vn : Build.MANUFACTURER check, network operator name check
|-> compiler : dx (possible dexmerge)
|-> manipulator : dexmerge
[*] /home/akib/Desktop/Thesis/Tools/APKiD/top-7/b41d8296242c6395eee9e5aa7b2c626a208a7acce979bc37f6cb7ec5e777665a.apk!classes.dex
|-> anti_vn : Build.MANUFACTURER check, Build.MODEL check, Build.PRODUCT check, Build.TAGS check, SIM operator check, network operator name check
|-> compiler : dx (possible dexmerge)
|-> manipulator : dexmerge
```

Figure 34: Compiler, Manipulator Details of Various Applications

This type of information can be useful to track an unknown application.

DroidLysis [47] is an extracting tool for Android application properties. It breaks down the Android program which the user gives automatically and searches the package or pulls apart data for various features. This tool is used to find the details of our application. Figure- 35 shows different properties that are extracted from the application that is used for training.

```

===== Report =====
Sanitized basename : vut.apk
SHA256             : e07579b8996a2360572d670e74fb9e1cf3b44bee53f4338c3208f50004118b94
File size          : 2019576 bytes
Is small           : False
Nb of classes      : 14
Nb of dirs         : 4

Certificate properties
algo               : SHA256withRSA
serialno          : 333a0b9b
country           : 500
owner              : C=debugging
timestamp         : (2022, 4, 18, 23, 55, 44)
year              : 2022

Manifest properties
activities         : ["com.ternuxhackers.id.MainActivity"]
main_activity     : com.ternuxhackers.id.MainActivity
package_name      : com.ternuxhackers.id
permissions       : ['SYSTEM_ALERT_WINDOW', 'RECEIVE_BOOT_COMPLETED', 'SET_WALLPAPER', '
READ_EXTERNAL_STORAGE', 'WRITE_EXTERNAL_STORAGE', 'READ_CONTACTS', 'READ_SMS', 'ACCESS_FIN
E_LOCATION', 'WAKE_LOCK', 'INTERNET', 'REQUEST_INSTALL_PACKAGE', 'CAMERA']
receivers         : ["com.ternuxhackers.id.BootReceiver"]
services          : ["com.ternuxhackers.id.MyService"]

Small properties / What the Dalvik code does
execute_native    : True (Executes shell or native executables)
get_package_info  : True (Gets information on package)
logcat            : True (Inspects or manipulates system logs)
reflection        : True (Uses Java Reflection)

Wide properties / What Resources/Assets do

ARM properties / What native ARM libraries do

DEX properties / About classes.dex format
magic             : 35

Kit properties / Detected 3rd party SDKs
END

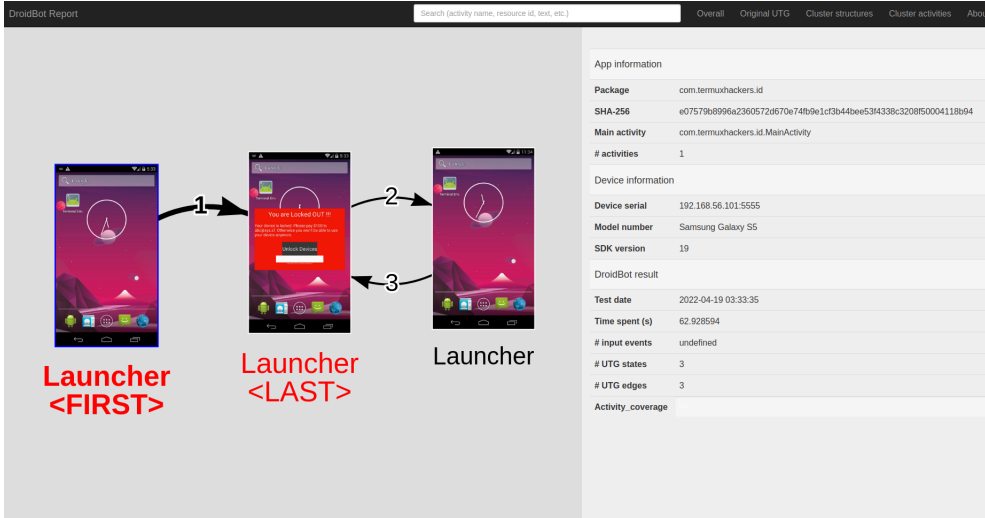
```

Figure 35: Properties Analysis Report of Application

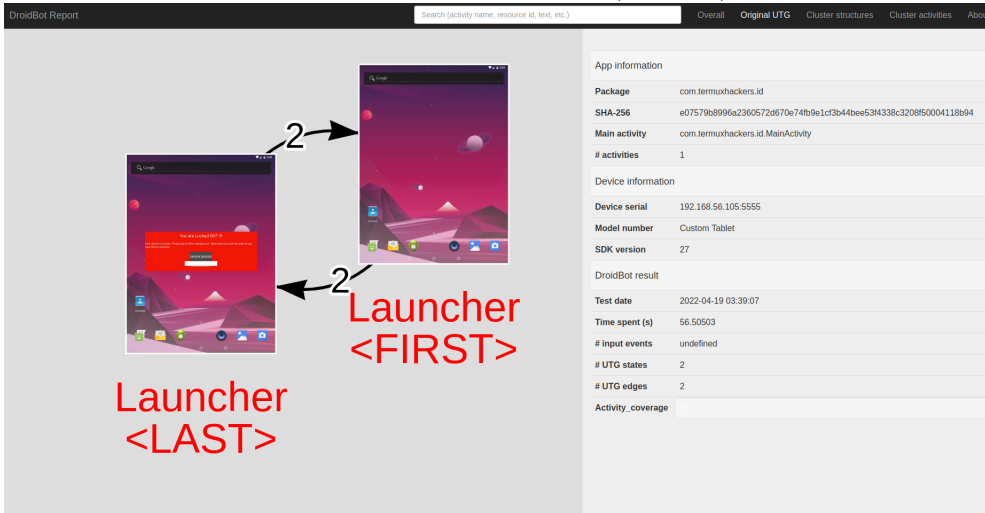
In the next part, the study of the same application was done using the droidbot [42] [43] tool. Here we tested our app in 3 different android versions and APIs. Figure- 36 show the outcomes of providing random or pre-identified input events to an app. It shows test coverage faster and creates a UTG also known as User-Interface Transition Graph after completing testing.



### a. UTG in Android 4.4 (API 19)



### b. UTG in Android 8.1 (API 27)



### c. UTG in Android 10 (API 29)

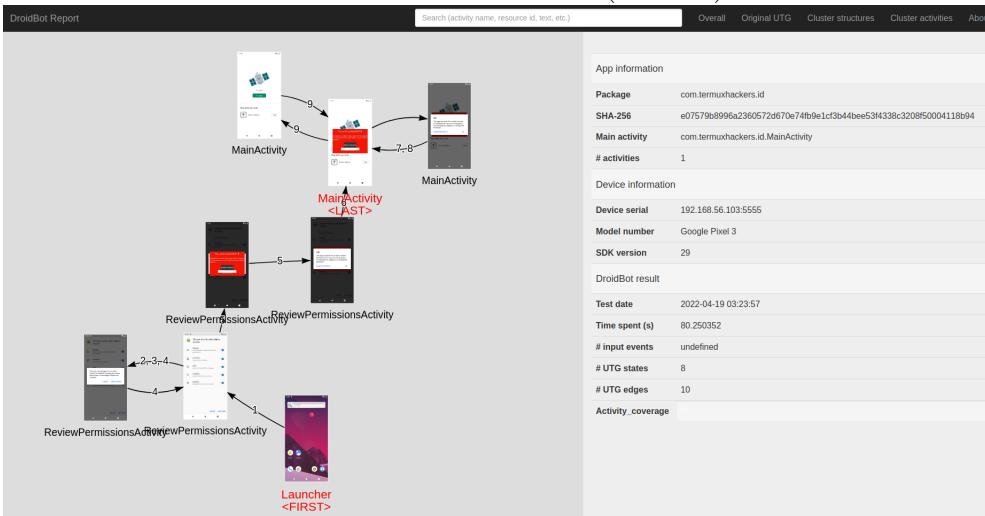


Figure 36: User-Interface Transition Diagram of Customized Application.

Package Name	SHA 256	Number of Security Vendors Marks This as Malicious
com.test.t000004	00a419a4ffe1819a51207b3140592734	31
com.bckalz.iphone5s	02f3ddaada005263619aebdb2d2a14e0	24
com.android.bluetooth	8b9520ef4054640badcef491213fbefe	1
fr.mildlyusefulsoftware.awesomesportcars	011889b7c5892e514b1c833d6bd4c476	1
com.android.browser	8753cc13bc19e2c36ecbd1ff6dbd26e0	1
com.cyanogenmod.filemanager	fb88a628def2cd0c1670aac20dd5191	1
com.mwr.dz	6e6ba57a704c5a0895ac9a152d4cc399	1
com.genymotion.superuser	7b5d2f0140cd5fb82e23909bfcdf45f1	1
com.android.camera	61fef546c6a4cd2f15fe50496fd5d54	1

Table 5: Analysis of Packages Reported as Malicious (Android 4.4, API 19)

Check All APK's [58] is a combination of scripts that use Drozer [56][57] and the VirusTotal API [60] to determine whether a phone is running malware-infected apps. This is helpful in violation of security when an analyst must distinguish between hundreds of valid apps. The test we conducted was among three android versions with different API levels. The goal is to find the vulnerable application that may be present in the device as there are a lot of applications running.

Here the Table- 5 indicates the possible malicious packages that were present in the first device (Android Version - 4.4)

The following Table- 6 shows the malicious packages in the device having android version - 8.1:

Package Name	SHA 256	Number of Security Vendors Marks This as Malicious
com.android.security	a92301c54d8bcd96a9f405a313b375e7	35
com.antivirus.kav	c9e3af6a4429197c05c18408f9f287ee	34
com.android.htmlviewer	f6a0bf1e4ba649b55cbb475aa4fb76b9	1
com.android.companiondevicemanager	91d894070afc94c31041e818531c06c9	1
com.android.defcontainer	49e0dc8eb85e313e78d8d105d22443b2	1
com.android.egg	54c55eea8760ac7bceb56875717d212b	1
com.android.nfc	ee8f59e0c9bd7feb8c96402b254499cd	1
com.android.sharedstoragebackup	eb949eafb8740507354d9ef49b3dd386	1
com.android.webview	f50a520dd087eaf1c07fcee87644340d	1
com.android.inputmethod.latin	f5c164de710eab4058c31de4dd753509	1
com.android.managedprovisioning	d030578fefbd0b0456a0f3b399d90539	1
com.android.smspsh	2b8656be9320b6b4f1ce215d3c5bf2bc	1
com.amaze.filemanager	9978ce22098cd11af1d02e2b2ddf7f37	1
com.mwr.dz	6e6ba57a704c5a0895ac9a152d4cc399	1
com.android.vpndialogs	ede9c632d6d55916f2f89bbeeca74abc	1
com.android.wallpaperbackup	abc650dbe2d3789fb90ab4ac00eff821	1
com.android.bluetooth	0b34e36f340681166ed14e2037e1c9c2	1
com.android.captiveportallogin	9c10470b5536b8fc4404ff2ea572e91f	1

Table 6: Analysis of Packages Reported as Malicious (Android 8.1, API 27)

The following Table- 7 shows the malicious packages in the device having android version - 10:

Package Name	SHA 256	Number of Security Vendors Marks This as Malicious
org.slempto.service:	a2603254188da3d67e4da5452e0304a9	30
com.android.bluetooth	759f3530b7455c50d24066f132ab078e	1
com.android.phone	11702d97134124453407d65affdf9bad	1
com.android.email	1866a87f79f9141bccabf00484f95270	1
com.android.permissioncontroller	c58c73d3c6928aa1a2d43bd34620b4d8	1
...	...	...

Table 7: Partial Analysis of Packages Reported as Malicious (Android 10, API 29)

In short, in three different systems we have found interesting behaviors from different packages. Table- 8 shows the percentage of malicious packages found in the conducted experiment:

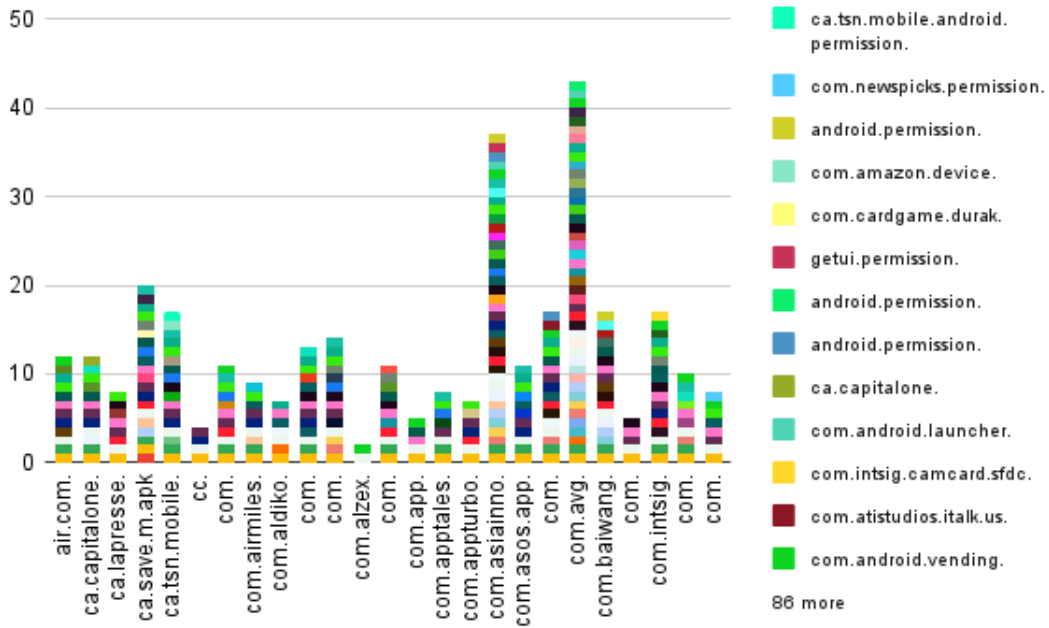
Android Version	Total Packages Analyzed	Malicious Packages Found	Percentage of Malicious Package
4.4 (API 19)	97	9	9.3 %
8.1 (API 27)	114	18	15.8 %
10 (API 29)	152	56	36.8 %

Table 8: Comparative Results of Malicious Packages in Different Version

In the PACE project [48] [49], the script will collect permission information from Malware and Benign software in their respective directories and combine it into a single Comma Separated Values file. ready to be fed into machine learning algorithms.

In Figure- 37, the overall picture of the gathered permission is presented. It can be seen that almost all the applications use a large number of permissions. From this information, it is very difficult to distinguish the malicious and benign applications. This dataset is then used in various machine learning algorithms. These algorithms will be used to create models. The better the training set the better results will be obtained.

a. Benign Applications Permission List Sample



b. Malicious Applications Permission List Sample

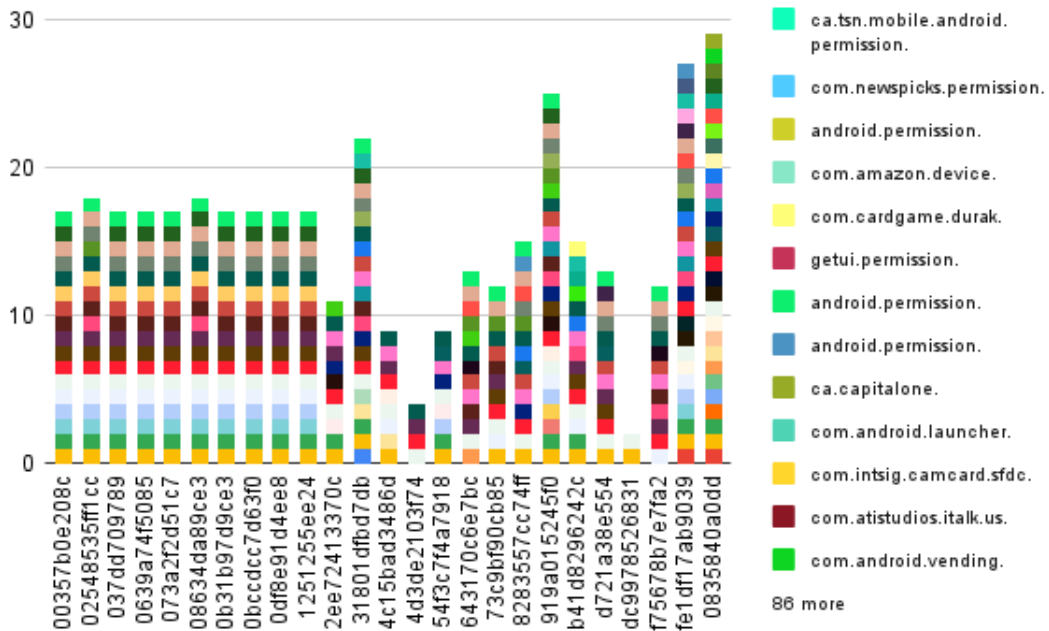


Figure 37: Pictoral Overview of the Permission List Gathered.

Before splitting the dataset into a classifier, first, some basic characteristics of both malware and benign applications are identified. The top 10 permissions extracted from the data that were used in both malware and benign are shown in figure 38. Here it is evident that most of the features used are similar in both malware and benign applications. Thus, to make a differentiation between them the machine needs to learn intelligently.

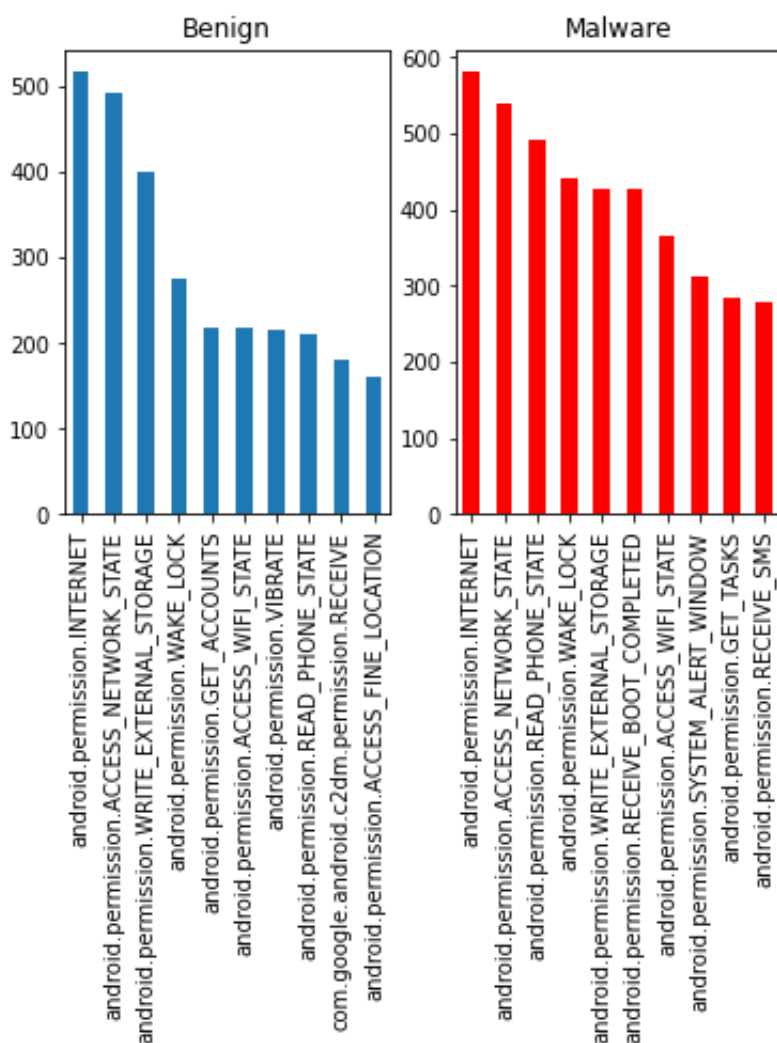


Figure 38: Top 10 Permissions Used (Benign vs Malware)

Here are some feature-based differences between benign and malware applications after considering the wider scenario. In figure- 39, some of the important features distinction is presented via graph. Some of the images clearly show that malware

is sending more permission requests for sensitive data than benign applications. “android.permission.READ\_SMS”, “android.permission.RECEIVE\_SMS”, “android.permission.READ\_CONTACTS”, “android.permission.CONTROL\_LOCATION\_UPDATES” are some examples of the extracted sensitive features that have a significantly higher access rate for malicious applications.

Table- 9 shows different algorithms showing accuracies gathered from the dataset used.

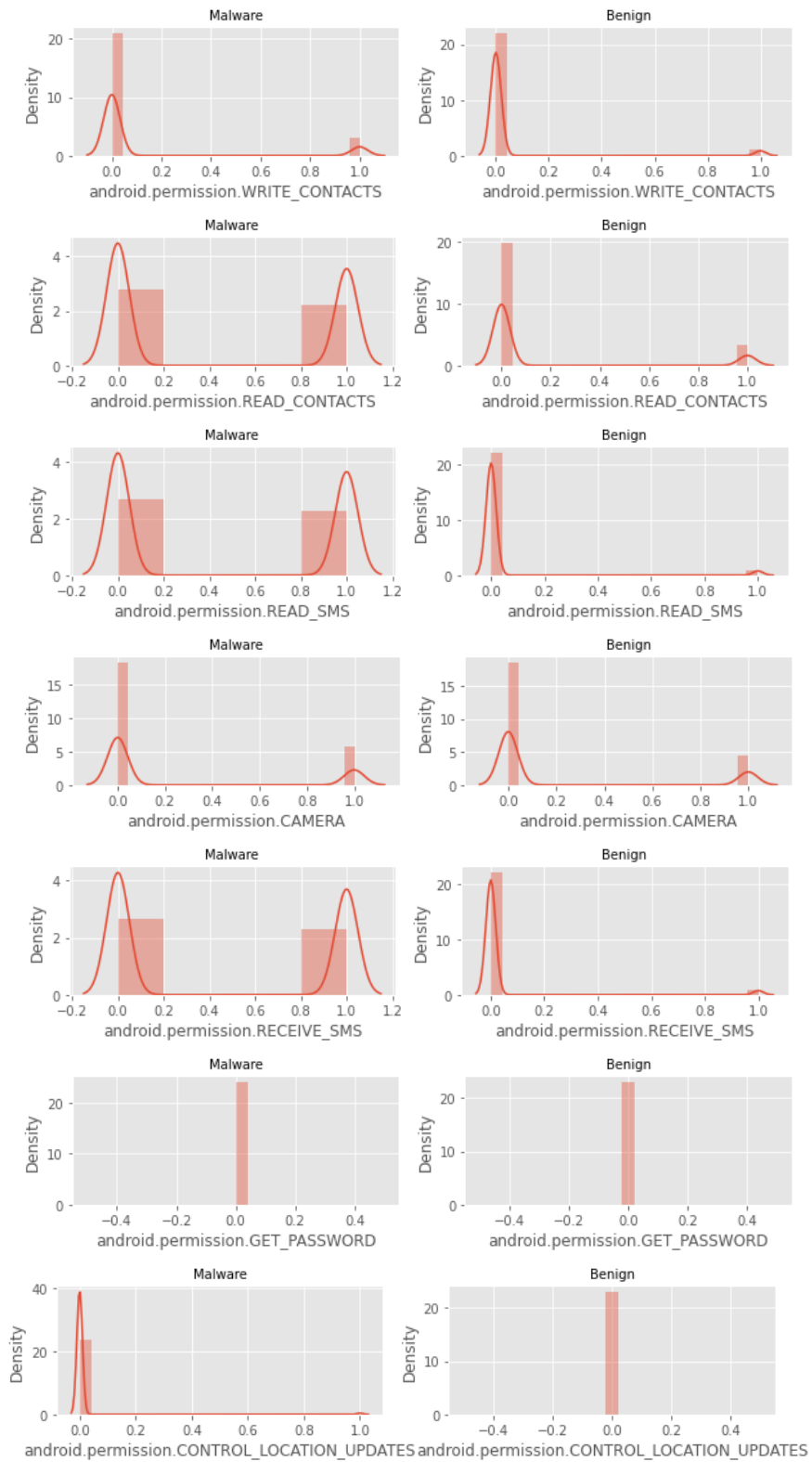


Figure 39: Permissions (Malware vs Benign)



Algorithm	Accuracy
Naive Bayes Algorithm	65 %
K-Nearest Neighbors:	
kneighbors 3	86 %
kneighbors 6	85 %
kneighbors 9	85 %
kneighbors 12	82 %
Decision Tree Algorithm	84 %
Random Forests Algorithm	89 %
TensorFlow Decision Forests Algorithm	90 %

Table 9: Different machine learning algorithm accuracy

This table shows that the TensorFlow Decision Forests Algorithm outperformed other existing algorithms while permission-based features are considered for malware analysis. Figure- 40 shows the model for TensorFlow Serving. This model indicates the most important feature as its root by considering this as a tree structure. Other nodes follow their own precedence.

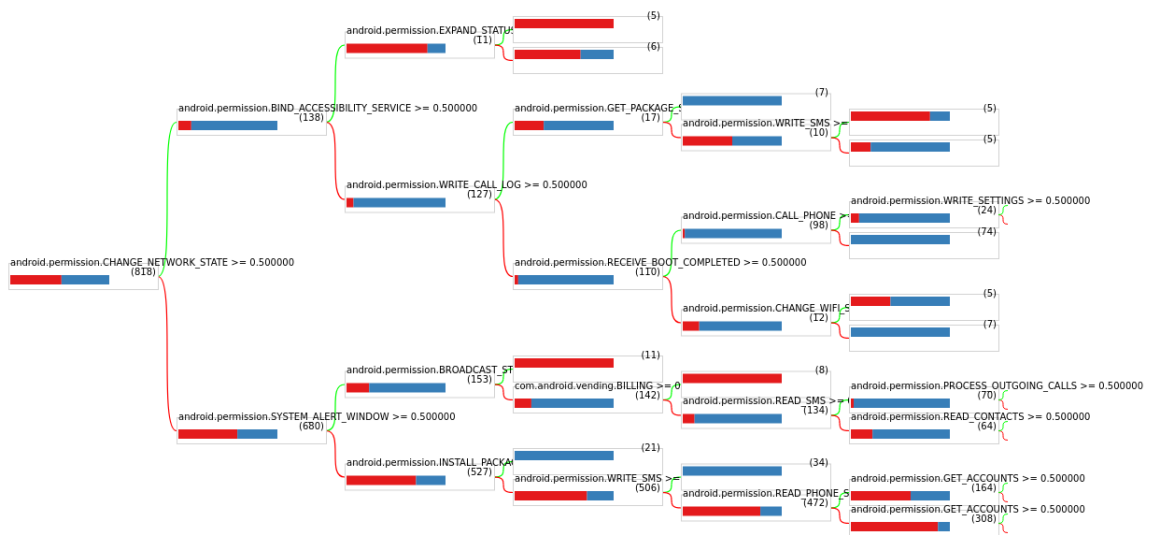


Figure 40: Model for TensorFlow Serving.

In Figure- 41, two graphs show accuracy rates and data loss rates over the number of trees. It is seen that as the number of trees grows the accuracy rates become more mature based on the fed data. Moreover, the loss rate of data tends to zero as the tree numbers increase.

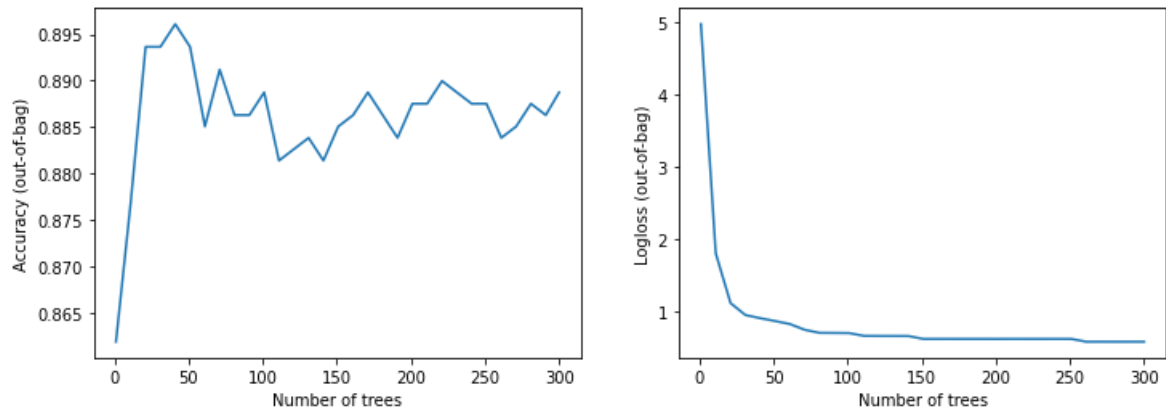


Figure 41: Accuracy and Data Loss Graph of TensorFlow Decision Forest Algorithm

## 8 Limitations and Challenges of The Conducted Work

### 8.1 Limitations

This research work is done with a very limited number of data. The result obtained from these data cannot be generalized to conclude for the whole malware world. Most of the analysis tools are version dependent. For a non-technical specialist background person, these are hard to use and understand. Also, the permission analysis was conducted to produce various types of results. Thus, these techniques cannot be unified and made as a whole unit to work together. Moreover, the collected application executables could not be tested one by one. Therefore, there can be questions about the application being malware or benign. In the work, the applications were collected from a small number of sources. There are numerous open-source as well as private (but accessible with permission) present which can bring significant results to this research work. As the models created were trained with predefined malware and benign samples, the results can be biased based on the dataset's behavior. Furthermore, the number of studied features is limited. This specific work eyes on the permission-based extraction data of the applications. It is clear that only permission analysis cannot study the whole characteristics of malware spreading. There can be many other features for which malware attacks occur.

### 8.2 Challenges

The experiment conducted on real users was very time-consuming. Also approaching the users and getting their feedback was difficult. Malware analysis studies have a lot of fields to cover. Finding a specific path and discovering something is not easy. As the world is more reliant on the internet, there is a lot of information available regarding open-source malware and its analysis techniques. Among those huge numbers of data, reliable data are quite hard to distinguish. Then

the techniques are also very tough to use and analyze. By setting proper environments and rules, these techniques can give the desired output. Many tools are also unmaintained. These tools are developed long ago and are currently unusable with the latest systems. The dataset gathering and storing is burdensome often-times. Even the data extraction can be hard to follow if one data is untraceable and unreadable. There are many occasions when the extraction method stopped working and started from the beginning as there was an internal error in the particular application. In addition, extracting the data is done in one go as different executions will provide different feature sets, which are quite hard to combine.

## 9 Conclusion and Future Work

It is still not possible to prevent all sorts of malware. There are many types of malicious programs or software that have not been identified or detected. There exists a concept of a zero-day vulnerability. An instance where attackers take advantage of a software security loophole to carry out an invasion is known as a "zero-day exploit." More dangerously, it is only known to the attackers, which means the developers have no idea such a loophole even exists. So detecting malware, preventing cyberattacks, and ensuring security is crucial in this age of information technology. This paper contains an up-to-date validation of malware analysis tools. Various aspects of malware analysis have been discussed including new malware attack patterns and datasets. Only permission extraction is not sufficient for malware detection. This paper contains a new dataset on which the latest ML algorithms have been run. With an accuracy of 90% TensorFlow decision forest algorithm has provided a significantly better performance over other machine learning algorithms such as the Naive Bayes Algorithm, K-Nearest Neighbor Algorithm, Decision Tree Algorithm, and Random Forest Algorithm after analysis of 1168 different applications. Taking permission to carry out specific actions on android devices into consideration, this dissertation concludes the TensorFlow decision forest tree as the best in classifying data.

There is a lot of work still needs to be done in the field of malware detection and analysis. The Android malware detection techniques should be updated with every new update of the system. There are a lot of open-source resources still available to explore. New and improved methods of data extraction can give a significant result that can contribute more to the analysis of malware. Additionally, more user-friendly and usable techniques can be introduced to stop the spread of malicious software. Machine learning-based algorithms, deep neural networks, and artificial intelligence can be used to predict malware more effectively and efficiently.

## References

- [1] Schmidt, A.D., Clausen, J.H., Camtepe, A. and Albayrak, S., 2009, October. Detecting symbian os malware through static function call analysis. In 2009 4th International Conference on Malicious and Unwanted Software (MALWARE) (pp. 15-22). IEEE.
- [2] Upchurch, J. and Zhou, X., 2016, October. Malware provenance: code reuse detection in malicious software at scale. In 2016 11th International Conference on Malicious and Unwanted Software (MALWARE) (pp. 1-9). IEEE.
- [3] Alashjaee, A.M. and Haney, M., 2021, January. Forensic Requirements Specification for Mobile Device Malware Forensic Models. In 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC) (pp. 0930-0935). IEEE.
- [4] Qiao, Y., Yun, X. and Zhang, Y., 2016, August. How to automatically identify the homology of different malware. In 2016 IEEE Trustcom/BigDataSE/ISPA (pp. 929-936). IEEE.
- [5] Barabosch, T. and Gerhards-Padilla, E., 2014, October. Host-based code injection attacks: A popular technique used by malware. In 2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE) (pp. 8-17). IEEE.
- [6] Zhao, B. and Lao, Y., 2018, October. Resilience of pruned neural network against poisoning attack. In 2018 13th International Conference on Malicious and Unwanted Software (MALWARE) (pp. 78-83). IEEE.
- [7] Ray, A. and Nath, A., 2016. Introduction to Malware and Malware Analysis: A brief overview. *International Journal*, 4(10).
- [8] Subrahmanian, V.S., Ovelgönne, M., Dumitras, T. and Prakash, B.A., 2015. Types of malware and malware distribution strategies. In *The Global Cyber-Vulnerability Report* (pp. 33-46). Springer, Cham.

- [9] Skoudis, E. and Zeltser, L., 2004. Malware: Fighting malicious code. Prentice Hall Professional.
- [10] Mishra, U., 2010. An introduction to computer viruses. Available at SSRN 1916631.
- [11] Apple, R. and Arch, C., 2007. Malicious Software-A Brief History.
- [12] Panko, R.R., 2003. Slammer: The first blitz worm. Communications of the Association for Information Systems, 11(1), p.12.
- [13] Rajesh, B., Reddy, Y.J. and Reddy, B.D.K., 2015. A survey paper on malicious computer worms. International Journal of Advanced Research in Computer Science and Technology, 3(2), pp.161-167.
- [14] Martin, J.C., Burge III, L.L., Gill, J.I., Washington, A.N. and Alfred, M., 2010. Modelling the spread of mobile malware. International Journal of Computer Aided Engineering and Technology, 2(1), pp.3-14.
- [15] J. Koret and E. Bachaalany, The antivirus hacker's handbook. Indianapolis, IN: John Wiley Sons Inc, 2015.
- [16] Team, U.S.S., 2010. Zeus malware: Threat banking industry.
- [17] J. Aycock, Spyware and Adware, vol. 50. Boston, MA: Springer US, 2011.
- [18] Landesman, Mary Landesman. "The First 25 Years Of Malware." Lifewire. [www.lifewire.com](https://www.lifewire.com/brief-history-of-malware-153616), March 9, 2021. <https://www.lifewire.com/brief-history-of-malware-153616>.
- [19] Ali, A., 2017. Ransomware: A research and a personal case study of dealing with this nasty malware. Issues in Informing Science and Information Technology, 14, pp.87-99.
- [20] Chess, B. and McGraw, G., 2004. Static analysis for security. IEEE security privacy, 2(6), pp.76-79.

- [21] Shijo, P.V. and Salim, A.J.P.C.S., 2015. Integrated static and dynamic analysis for malware detection. *Procedia Computer Science*, 46, pp.804-811.
- [22] C. H. Malin, E. Casey, and J. M. Aquilina, *Malware Forensics: Investigating and Analyzing Malicious Code*. Syngress, 2008.
- [23] Talukder, S., 2020. Tools and techniques for malware detection and analysis. arXiv preprint arXiv:2002.06819.
- [24] Landage, Jyoti, and M. P. Wankhade. "Malware and malware detection techniques: A survey." *International Journal of Engineering Research and Technology (IJERT)* 2.12 (2013): 2278-0181.
- [25] Robiah, Y., et al. "A new generic taxonomy on hybrid malware detection technique." arXiv preprint arXiv: 0909.4860 (2009).
- [26] Tahir, R., 2018. A study on malware and malware detection techniques. *International Journal of Education and Management Engineering*, 8(2), p.20.
- [27] Talal, M., Zaidan, A.A., Zaidan, B.B., Albahri, O.S., Alsalem, M.A., Albahri, A.S., Alamoodi, A.H., Kiah, M.L.M., Jumaah, F.M. and Alaa, M., 2019. Comprehensive review and analysis of anti-malware apps for smartphones. *Telecommunication Systems*, 72(2), pp.285-337.
- [28] Fazeen, M., Dantu, R. (2014). Another free app: Does it have the right intentions? In 2014 twelfth annual international conference on privacy, security and trust (PST) (pp. 282–289)
- [29] Yerima, S.Y., Sezer, S. and Muttik, I., 2015. High accuracy android malware detection using ensemble learning. *IET Information Security*, 9(6), pp.313-320.
- [30] M. Chandramohan and H. B. K. Tan, "Detection of Mobile Malware in the Wild," *Computer*, vol. 45, no. 9. pp. 65–71, 2012.
- [31] Mell, P., Kent, K. and Nusbaum, J., 2005. Guide to malware incident prevention and handling (pp. 800-83). Gaithersburg, Maryland: US Department of



Commerce, Technology Administration, National Institute of Standards and Technology.

- [32] Jyoti Landage, Prof. M. P. Wankhade, 2013, Malware and Malware Detection Techniques : A Survey, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH TECHNOLOGY (IJERT) Volume 02, Issue 12 (December 2013),
- [33] Fuchsberger, A., 2005. Intrusion detection systems and intrusion prevention systems. Information Security Technical Report, 10(3), pp.134-139.
- [34] Qiu, L., Varghese, G. and Suri, S., 2001, June. Fast firewall implementations for software-based and hardware-based routers. In Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (pp. 344-345).
- [35] Schneider, F.B., 2003. Least privilege and more [computer security]. IEEE Security Privacy, 1(5), pp.55-59.
- [36] Cavusoglu, H., Cavusoglu, H. and Zhang, J., 2008. Security patch management: Share the burden or share the damage?. Management Science, 54(4), pp.657-670.
- [37] Termuxhackers-Id. (n.d.). Termuxhackers-ID/sara: SARA - simple Android ransomware attack. GitHub. Retrieved April 18, 2022, from <https://github.com/termuxhackers-id/SARA>
- [38] Laya Taheri, Andi Fitriah Abdulkadir, Arash Habibi Lashkari; Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls, The IEEE (53rd) International Carnahan Conference on Security Technology, India, 2019
- [39] Kiss, N., Lalande, J.F., Leslous, M., and Viet Triem Tong, V. 2016. Kharon dataset: Android malware under a microscope. In Learning from Authoritative Security Experiment Results. The USENIX Association.

- [40] Malware Sample Exchange. MalwareBazaar. (n.d.). Retrieved April 23, 2022, from <https://bazaar.abuse.ch/>
- [41] Quark-Engine. (n.d.). Quark-engine/quark-engine: Android malware (analysis: Scoring) system. GitHub. Retrieved April 18, 2022, from <https://github.com/quark-engine/quark-engine>
- [42] Li, Y., Yang, Z., Guo, Y., and Chen, X. 2017. DroidBot: A Lightweight UI-Guided Test Input Generator for Android. In Proceedings of the 39th International Conference on Software Engineering Companion (pp. 23–26). IEEE Press.
- [43] HoneyNet. (n.d.). HoneyNet/droidbot: A lightweight test input generator for Android. similar to monkey, but with more intelligence and cool features! GitHub. Retrieved April 18, 2022, from <https://github.com/honeyNet/droidbot>
- [44] MLDroid. (n.d.). MLDroid/Androwarn: Yet another static code analyzer for malicious Android Applications. GitHub. Retrieved April 18, 2022, from <https://github.com/MLDroid/androwarn>
- [45] Hexabin. (n.d.). Hexabin/APKSTAT: Automated Information Retrieval from APKs for initial analysis. GitHub. Retrieved April 18, 2022, from <https://github.com/hexabin/APKStat>
- [46] Rednaga. (n.d.). Rednaga/apkid: Android Application Identifier for Packers, protectors, obfuscators and oddities - peid for Android. GitHub. Retrieved April 18, 2022, from <https://github.com/rednaga/APKiD>
- [47] Cryptax. (n.d.). Cryptax/droidlysis: Property extractor for Android apps. GitHub. Retrieved April 18, 2022, from <https://github.com/cryptax/droidlysis>
- [48] A. Kumar, V. Agarwal, S. K. Shandilya, A. Shalaginov, S. Upadhyay, B. Yadav (2019). PACE: Platform for Android Malware Classification and Performance Evaluation. In 2019 IEEE International Conference on Big Data (Big Data) (pp. 4280-4288).

- [49] Ajit Kumar, K.S. Kuppusamy, G. Aghila (2018). FAMOUS: Forensic Analysis of MOBILE devices Using Scoring of application permissions. *Future Generation Computer Systems*, 83, 158-172.
- [50] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.E.R.T., 2014, February. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss* (Vol. 14, pp. 23-26).
- [51] MLDroid. (n.d.). MLDroid/Drebin: Drebin - NDSS 2014 re-implementation. GitHub. Retrieved April 18, 2022, from <https://github.com/MLDroid/drebin>
- [52] Allix, K., Bissyandé, T.F., Jérôme, Q., Klein, J. and Le Traon, Y., 2016. Empirical assessment of machine learning-based malware detectors for Android. *Empirical Software Engineering*, 21(1), pp.183-211.
- [53] MLDroid. (n.d.). MLDroid/csbd: The repository contains the python implementation of the android malware detection paper: "empirical assessment of machine learning-based malware detectors for Android: Measuring the gap between in-the-lab and in-the-wild validation scenarios". GitHub. Retrieved April 18, 2022, from <https://github.com/MLDroid/csbd>
- [54] Narayanan, A., Chandramohan, M., Chen, L. and Liu, Y., 2017. Context-aware, adaptive and scalable android malware detection through online learning (extended version). arXiv preprint arXiv:1706.00947.
- [55] Narayanan, A., Chandramohan, M., Chen, L. and Liu, Y., 2017. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3), pp.157-175.
- [56] FSecureLABS. (n.d.). FSecureLABS/drozer: The leading security assessment framework for Android. GitHub. Retrieved April 18, 2022, from <https://github.com/FSecureLABS/drozer>
- [57] inf0junki3. (2017, October 6). Checking your Android device for known malware. Kudelski Security Research. Retrieved April 19, 2022, from

<https://research.kudelskisecurity.com/2017/08/08/checking-your-android-device-for-known-malware/>

- [58] Kudelskisecurity. (n.d.). Kudelskisecurity/check\_all\_apks: Check all APK's – scripts for checking your phone for malware. GitHub. Retrieved April 19, 2022, from [https://github.com/kudelskisecurity/check\\_all\\_apks](https://github.com/kudelskisecurity/check_all_apks)
- [59] Xaviha. (n.d.). Xaviha/Stormbreaker: Tool Social Engineering [Access Webcam Microphone OS Password Grabber Location Finder] with Ngrok. GitHub. Retrieved April 25, 2022, from <https://github.com/xaviha/stormbreaker>
- [60] Virustotal. (n.d.). Retrieved April 25, 2022, from <https://www.virustotal.com/gui/home/upload>
- [61] Herron, N., Glisson, W.B., McDonald, J.T. and Benton, R.K., 2021, January. Machine learning-based android malware detection using manifest permissions. Proceedings of the 54th Hawaii International Conference on System Sciences.
- [62] Hahn, S., Protsenko, M. and Müller, T., 2016. Comparative evaluation of machine learning-based malware detection on android. Sicherheit 2016-Sicherheit, Schutz und Zuverlässigkeit.
- [63] Anderson, H.S. and Roth, P., 2018. Ember: an open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637.
- [64] Sewak, M., Sahay, S.K. and Rathore, H., 2018, August. An investigation of a deep learning-based malware detection system. In Proceedings of the 13th International Conference on Availability, Reliability, and Security (pp. 1-5).
- [65] Joyce, R.J., Amlani, D., Nicholas, C. and Raff, E., 2021. MOTIF: A Large Malware Reference Dataset with Ground Truth Family Labels. arXiv preprint arXiv:2111.15031.

- [66] Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W. and Ye, H., 2018. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7), pp.3216-3225.
- [67] Mahindru, A. and Singh, P., 2017, February. Dynamic permissions based android malware detection using machine learning techniques. In *Proceedings of the 10th innovations in software engineering conference* (pp. 202-210).
- [68] Arslan, R.S., Dođru, İ.A. and Barişçi, N., 2019. Permission-based malware detection system for android using machine learning techniques. *International journal of software engineering and knowledge engineering*, 29(01), pp.43-61.
- [69] Milosevic, N., Dehghantanha, A. and Choo, K.K.R., 2017. Machine learning aided Android malware classification. *Computers Electrical Engineering*, 61, pp.266-274.