REPUBLIC OF TURKEY
ADANA ALPARSLAN TÜRKEŞ SCIENCE AND TECHNOLOGY UNIVERSITY

GRADUATE SCHOOL OF INFORMATICS
DEPARTMENT OF CYBER SECURITY

DETECTING ANDROID MALWARE BY USING FUZZY SET-BASED WEIGHTING METHOD AND FIREFLY OPTIMIZATION ALGORITHM

VAHİDE NİDA UZEL

MASTER OF SCIENCE

ADANA 2020

REPUBLIC OF TURKEY
ADANA ALPARSLAN TÜRKEŞ SCIENCE AND TECHNOLOGY UNIVERSITY

GRADUATE SCHOOL OF INFORMATICS
DEPARTMENT OF CYBER SECURITY

DETECTING ANDROID MALWARE BY USING FUZZY SET-BASED WEIGHTING METHOD AND FIREFLY OPTIMIZATION ALGORITHM

VAHİDE NİDA UZEL
MASTER OF SCIENCE

SUPERVISOR
ASST. PROF. DR. ESRA SARAÇ EŞSİZ

ADANA 2020

DETECTING ANDROID MALWARE BY USING FUZZY SET-BASED WEIGHTING METHOD AND FIREFLY OPTIMIZATION ALGORITHM

submitted by **Vahide Nida UZEL** in partial fulfillment of the requirements for the degree of **Master of Science in Department of Cyber Security, Adana Alparslan Türkeş Science and Technology University** by,

Assoc. Prof. Dr. Serdar YILDIRIM
Graduate School Director

Asst. Prof. Dr. Ali İNAN
Department Chair

Asst. Prof. Dr. Esra SARAÇ EŞSİZ
Supervisor

Thesis Committee

Asst. Prof. Dr. Esra SARAÇ EŞSİZ
Adana Alparslan Türkeş Science and Technology University

Prof. Dr. Selma Ayşe ÖZEL
Çukurova University

Asst. Prof. Dr. Ali İNAN
Adana Alparslan Türkeş Science and Technology University

Thesis Defense Date

10/01/2020

I hereby declare that all information in this thesis has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all information that is not original to this work.

Vahide Nida UZEL

ABSTRACT

DETECTING ANDROID MALWARE BY USING FUZZY SET-BASED WEIGHTING METHOD AND FIREFLY OPTIMIZATION ALGORITHM

Vahide Nida UZEL

Department of Cyber Security

Supervisor: Asst. Prof. Dr. Esra SARAÇ EŞSİZ

January 2020, 55 pages

Android OS is open-source and easy to use mobile operating system. It is also user-friendly with many other features. In this way, it is a very preferred operating system on mobile phones. As a result, it becomes the target of malicious people. Applications installed on the Android operating system from the Google Play Store or by third-party application providers, also known as Android package files, may contain malicious software. So far, a variety of analyzes and detections have been made to detect such malware. While detecting malware, good results have been obtained with various methods, but malicious people have developed methods of hiding themselves against these methods.

We propose a new feature selection method based on Firefly Optimization Algorithm with the Fuzzy Set-Based weighting method. The proposed method performs better than traditional feature selection methods with fewer features. The experimental results of this study proved that Firefly Optimization is an acceptable optimization algorithm for feature selection to detect malware in terms of classification performance and classification runtime. In addition, experimental evaluation of TF-IDF and Fuzzy Set-Based weighting methods indicates the effectiveness of the Fuzzy Set-Based weighting with a full feature set.

Keywords: Malware Detection, Android Operating System, Fuzzy Set-Based Weighting, Firefly Optimization Algorithm, Machine Learning, Feature Selection, Data Mining, Nature Inspired Algorithm

ÖZET

BULANIK KÜME TABANLI AĞIRLIKLANDIRMA YÖNTEMİNİ VE ATEŞBÖCEĞİ OPTİMİZASYON ALGORİTMASINI KULLANARAK ANDROID TABANLI KÖTÜCÜL YAZILIMLARI TESPİT ETME

Vahide Nida UZEL

Siber Güvenlik Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Esra SARAÇ EŞSİZ

Ocak 2020, 55 sayfa

Android işletim sistemi açık kaynak kodlu olması, kolay kullanılabilir olması ve diğer birçok özelliği ile kullanıcıya hitap etmesi sayesinde telefonlarda çok tercih edilen bir işletim sistemi haline gelmiştir. Bunun bir sonucu olarak, kötü niyetli insanların hedefi olmuştur. Android işletim sistemine Google Play Store'dan veya 3.parti uygulama sağlayıcılar tarafından yüklenen uygulamalar, diğer bir adıyla Android paket dosyaları, içerisinde kötücül yazılımlar ile birlikte gelmeye başlamıştır. Bu tür kötücül yazılımları tespit etmek için bu zamana kadar çok çeşitli analizler ve tespitler yapılmıştır. Kötücül yazılımları tespit ederken çeşitli yöntemler ile güzel sonuçlar alınmış olsa bile, tespit etme yöntemleri geliştikçe kötücül yazılımların kendilerini gizleme yöntemleri de aynı oranda artmaya başlamıştır.

Bu çalışmada Ateş Böceği Optimizasyon algoritması tabanlı yeni bir nitelik seçim yöntemi önerilmiştir. Buna ek olarak sıklıkla kullanılan ağırlıklandırma yöntemi TF-IDF ağırlıklandırmanın yanı sıra Bulanık Küme Tabanlı ağırlıklandırma yöntemi de kullanılarak sonuçları kıyaslanmıştır. Yapılan testler önerilen nitelik seçimi yönteminin geleneksel nitelik seçim yöntemlerine göre daha iyi sonuçlar verdiğini göstermiştir. Alınan sonuçlara göre Ateş Böceği Optimizasyon algoritması sınınflama performansı ve sınıflama süresi anlamında kötücül yazılım tespitinde kabul edilebilir bir algoritmadır. Buna ek olarak, TF-IDF ve Bulanık Küme Tabanlı ağırlıklandırma yöntemlerinin deneyleri Bulanık Küme Tabanlı ağırlıklandırma yönteminin nitelik seçimi yapılmadığı durumlarda TF-IDF ağırlıklandırma yönteminden daha iyi sonuçlar verdiğini göstermiştir.

Anahtar Kelimeler: Kötücül Yazılım Tespiti, Android İşletim Sistemi, Bulanık Küme Tabanlı Ağırlıklandırma, Ateş Böceği Optimizasyon Algoritması, Makine Öğrenmesi, Özellik Seçimi, Veri Madenciliği, Doğa Esinli Algoritmalar

To my dear family who always supports me...

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude my invaluable advisor Asst. Prof. Dr. Esra SARAÇ EŞSİZ who has always supported me with her encouraging and directing attitude and enlightened me with her knowledge and experience during my graduate education.

I would like to express my gratitude to the precious Prof. Dr. Selma Ayşe ÖZEL who shared her vast knowledge and experience with me while I was studying for a master's degree at Çukurova University.

I would like to thank my committee members Prof. Dr. Selma Ayşe ÖZEL and Asst. Prof. Dr. Ali İNAN for their suggestions and contributions.

I would like to thank my dear fiancé and colleague Res. Asst. Yasir KILIÇ for helping me to evaluate the events from a different perspective and broaden my horizons.

I would like to express my gratitude to my beloved parents and dear brother who supported me in all matters and are always with me on my good and bad days.

Finally, I would like to thank all my teachers and friends who contributed to me throughout my education life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

HAL	: Hardware Abstraction Layer
ART	: Android Runtime
JVM	: Java Virtual Machine
DVM	: Dalvik Virtual Machine
JIT	: Just-in-time
AOT	: Ahead-of-time
APKs	: Android Package Files
NDK	: Native Development Kit
XML	: Extensible Markup Language
CFG	: Control Flow Graph
API	: Application Programming Interface
AA	: Apriori Algorithm
PSO	: Particle Swarm Optimization
CMAR	: Classification-based Multiple Association Rule
BC	: Bayesian Classifier
SVM	: Support Vector Machine
B	: Bytes
BB	: Bytes Basic
BBA	: Bytes Basic Assembly
MAIL	: Malware Analysis Intermediate Language
FCBF	: Fast Correlation-based Filter
OFEFLY	: Optimized Feature Selection using Fuzzy Entropy and Firefly
ANFIS	: Adaptive Neuro Fuzzy Inference System
FCM	: Fuzzy C-Means Clustering
RMSE	: Root Mean Squared Error
CTE	: Cyber Terror and Extremism
MLP	: Multilayer Perceptron
FOA	: Firefly Optimization Algorithm
JI	: Jaccard Index

JIs : Jaccard Indexes

FSB : Fuzzy Set-Based

FFS : Firefly Feature Selection

TF : Term Frequency

IDF : Inverse Document Frequency

KNN : K-Nearest Neighbor

RF : Random Forest

RT : Random Tree

RBFN : Radial Basis Function Networks

NB : Naïve Bayes

Y_d : Input Vector

o_{di} : Output Vector of ith Hidden Layer

c_i : Center of ith Hidden Layer

H : Nearest Hit

M : Nearest Miss

o_i : Observed Value

e_i : Expected Value

β_0 : Initial Light Value

γ : Absorption Coefficient

1. INTRODUCTION

With the development of technology, many innovations entered our lives. One of these innovations is undoubtedly smartphones. In this age, smartphones have become indispensable. We can handle our daily routine with the help of smartphones. We can give an example of such routines as banking transactions, social media accounts, and communication networks. Although there are many benefits, the information shared in the virtual world can be completely reversed as a result of the seizure by malicious people. Therefore, it is essential to take advantage of technology and keep security at the highest level.

It wouldn't be wrong to say that smarthphones are small computers. There are a total of 2.71 billion smartphone users in the world. This figure is 35.13% of the world's population (Turner, 2019). Smartphones have an operating system just like computers. There are various operating systems such as BlackBerry, iOS, Windows Phone, and Android. The most preferred operating system is Android. The reasons for this, it is easy to understand, user-friendly and, most importantly, open-source and free. When Android first came out in 2008, iOS was the leading operating system. Android market share in 2010 was less than 5%. In 2013, however, Android's luck turned and it had 88% of market share in 3 years. This means that Android has surpassed the IOS operating system by a landslide. In 2016, there were 3.4 million apps in the Google Play Store and only 2.2 million apps in the app store. In 2017, its market share fell to 74%, but it still maintained its leading position. In 2018, the market share of Android in the world was determined as 72.23%. In 2022, this ratio is expected to rise up to 85.5% (G., 2019). Android is used by many people. This leads to malicious people turning their attention there. This has pushed the world of science to take extra care to protect the Android operating system. They have developed software to detect malware. In particular, the concept of Artificial Intelligence (AI) has recently entered into our lives. Thus, a good step has been taken in detecting malware and AI has been used in various studies.

Malware is generally detected by antivirus programs. The top 5 antivirus programs in 2019 are as follows: Avast Mobile Security, Bitdefender Antivirus Free, AVG, McAfee Security & Power Booster Free and Kaspersky Mobile Antivirus (Drake & Turner, 2019). Antivirus programs generally make a signature-based detection. This means that malware can be detected if the program matches the malware stored in the database. This type of detection

method is not a very healthy method because it cannot detect even the same malware with slightly changed content. In addition, the time limitation is very important when detecting malware. After running the program, the detection will not work because the malware will have already seized the phone. From here we need to pay attention to 2 things when detecting malware: one of them is to detect quickly and the other is to detect correctly.

As mentioned earlier, a good step has been taken thanks to AI to detect malware. However, the malware also develops against these methods. Therefore, different methods are needed.

In this section of the thesis, the background of the malware, malware detection techniques, analysis types and the aim of the thesis are presented.

1.1. Malware History and Types

Today, we use computer systems and the Internet in almost all areas of our lives. Computer systems and the Internet have many advantages, such as fast shopping, communicating with someone that is far away from you. In order to carry out such transactions, we may need to keep our personal information on the system and share it on the internet. Considering that malicious people may use these systems, they can access your personal information and steal your money from your account or crash your computer. These malicious people use malware to achieve their goals.

The term malware is an abbreviation of malicious software. It is specifically designed to harm, interfere or gain authorized access to a computer system. It has gone through many stages until it becomes harmful. First of all, two Pakistani brothers produced a virus with no harm. Then, Windows malware was formed. After the Internet became widespread, network malware becomes more active. Before 2010, ransomware was among the most dangerous malware. In 2010, a USB stick was secretly introduced into the Iranian nuclear program. As a result, the system is infected with a virus. This virus has corrupted the system. And the virus has set the system not to alert. Thus it seemed that there was no apparent problem, but it caused serious damage to the system. It was thought that those who wanted to cause this damage were secret agents of other countries. This event was later called Stuxnet. With the Stuxnet incident, it was understood that malware could be used for cross-country virtual espionage and sabotage. This is the kind of malware that is seen as the main threat today (Milošević, 2013).

1.1.1. Malware types according to their purpose

There are many kinds of malware. Some of them are known as bots, bugs, SMS malware, backdoor, adware, rootkits, spyware, trojan horses, viruses, and worms.

Bots (Saha & Gairola, 2005) which are the abbreviation of robots, are mostly used for emulating human behavior. They are actually created for great purposes like Google search engines and game bots. But attackers use bots for malicious intentions. They can install the software without user knowledge. Then they can remote control of the system and run attack commands.

Bugs (Soner, Soner, & Yaday, 2015) are errors in the program's source code or compiler. These errors may be too small to detect for a long time or they may be too big and can crash the computer. Security errors are the most dangerous ones. Because of these, attackers can pass user authentication stage, access privileges or steal data.

SMS malware sends unsolicited calls and messages without the user's permission and knowledge. Messages and calls are then forwarded to paid services to generate revenue from them. In this way, malware emitters can provide a large amount of money for themselves.

Backdoor is a vulnerability that occurs intentionally or unintentionally. It provides external access to the system. Thus, data can be added to the system or imported from the system. Backdoor has benefits but it is often harmful. Therefore, the backdoor should not be left in the system and should be closed, if any. An example of backdoor's benefit is that it helps to prevent children from accessing harmful sites. An example of its loss is that malicious people retrieve important data from the system.

Adware is software for displaying ads on a computer or phone. At the same time, this software collects and markets information such as which sites the user enters. Adware is harmless when it shows ads with the user’s permission, otherwise, it will be harmful.

Rootkits (Yim, Kim, Park, Lee, & You, 2012) gain root privileges on the system and they can hide previously installed malware on the system. And malware can run without any system alert.

Spyware (Gutzman, Sweep, & Tambo, 2003) includes all the technology tools used to collect information without the consent of the person such as collecting keystrokes, screen monitoring and data collection from account information.

In Ransomware (Tailor & Patel, 2017), malicious code infects the system and encrypt important data or lock the system. The purpose of the attacker to do this is to demand money for information.

Trojan Horse (Zhenfang, 2015) is a very dangerous malware. It can steal a file, system information or passwords by using the remote control of the system. Also, it can format the hardware.

Viruses (Anupriya, Nithya, & Ponmalar, 2018) are malware that can copy themselves. It spreads to the computer by connecting itself to another program.

Worms (Wang, Wen, Xiang, & Zhou, 2014) are a kind of viruses, but unlike viruses, they have the ability to replicate and spread themselves.

1.1.2. Malware types according to hiding methods

Encrypted malware (Uppal, Mehra, & Verma, 2014) encrypts the malware code to hide. The code will run when the decrypt algorithm runs and the encrypted code turns into executable and meaningful code. It uses a new encryption key for each encryption, but it is easy to find this type of malware because the decryption algorithm always stays the same.

The Oligomorphic (Sharma & Sahay, 2015) is the most advanced malware that is encrypted. In encrypted malware, the decryption algorithm always stays the same. But, in oligomorphic malware, there are several decryption algorithms. In oligomorphic malware, decryption is more difficult according to encrypted malware. But, it can still be detected because of the limited number of decryption algorithms. To detect with signature-based detection, each decrypted code version of the malware must be registered.

While oligomorphic malware produces a limited number of decryption algorithms, polymorphic malware (Sharma & Sahay, 2015) produces unlimited decryption algorithms. It has a mutation engine to produce unlimited decryption algorithms in it, and it uses obfuscation techniques. But a part of the code always remains the same so it can be detected by signature-based methods.

Metamorphic Malware (Sharma & Sahay, 2015) are similar to polymorphic malware, it uses obfuscation methods but does not use encryption algorithm. It changes the code in each iteration. Therefore, it is very difficult to detect with the signature-based method.

1.1.3. Obfuscation techniques

Malware uses various techniques to hide. Obfuscation is the general name of these techniques. Adding space character or comment line to the source code, or deleting space character or comment line from the source code is the simplest obfuscation method that shows the source code as a different code.

Changing variable names in the source code is another frequently used obfuscation method. Malware can also change the order of the codes, and as a result, there won't be any syntax error in the source code, but it won't be run correctly.

As a different obfuscation method, malware can replace the source code with another source code. Adding a new non-functional code in the source code is another obfuscation method that only caused confusion. Replacing the register of the program code by another unused register is also an obfuscation method.

All these hiding techniques change the signature of the code. Therefore, Signature-based and Behavior-based detection is insufficient to recognize these techniques. Heuristic-based methods have been developed to overcome such hiding techniques.

1.2. Android Structure

Android is more preferred and an open-source operating system for smartphones. It is easy to use and has a lot of applications on it.

Android consists of 4 different layers. These layers are the application layer, framework layer, middleware layer, and the kernel layer. These layers will be mentioned, respectively in Figure 1.1 (Android Developers, n.d.).

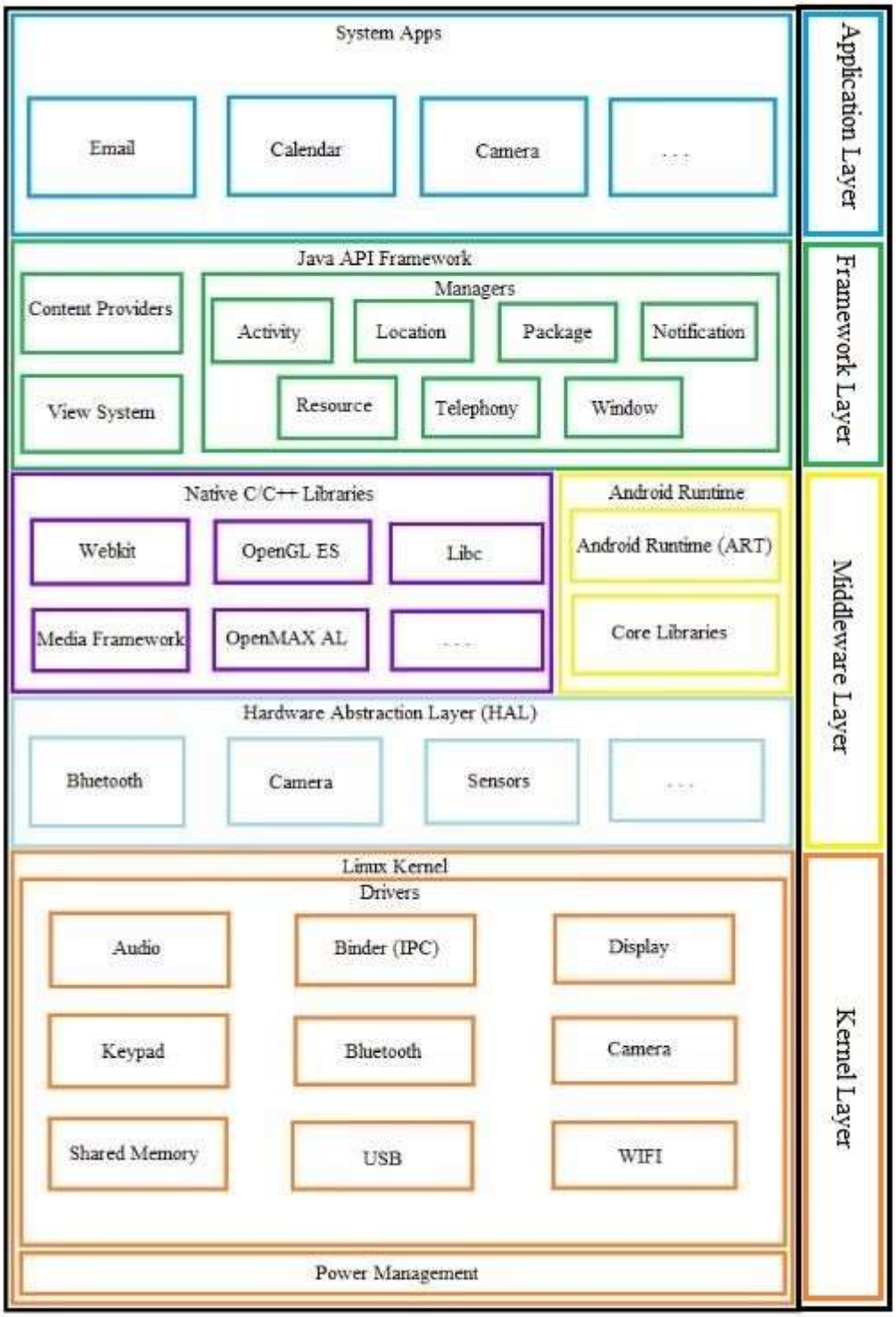

Figure 1.1. The Android Software Stack

The top layer is the application layer. System applications are pre-installed on a smartphone. There are also several applications that users can download from the Google Play Store according to their needs.

There is a framework layer on a sublayer of the application layer. This layer contains the basic functions that are necessary to run applications. For example, content providers are used to accessing applications with each other; the telephony manager is used for a voice call; the location manager which defines the geographical region of a phone. Without this layer, applications cannot communicate with each other, make calls or access information such as location.

Native libraries, android runtime, and Hardware Abstraction Layer (HAL) are in the middleware layer. Native libraries are libraries that are used to add features to the application from the outside. These libraries usually contain C/C++ languages. For example, the OpenGL library helps to draw 2D and 3D graphics in the application. Android runtime (ART) is similar to Java Virtual Machine (JVM) used in java language. JVM is a platform on which java works, whereas ART is a platform for android. ART has been used after version 4.0 of android. Before 4.0, a Dalvik Virtual Machine (DVM) was used. The difference is that DVM uses Just-in-time (JIT) compilation, while ART uses Ahead-Of-Time (AOT) compilation. In JIT, debugging occurs each time the application is opened, while in AOT the application is already debugged. This means that there is no need to debug each time the application is opened. This saves the application and phone speed and time. Hardware Abstraction Layer (HAL) acts as an intermediary between software and hardware. The software equipment communicates with HAL rather than directly with the hardware. HAL passes these requests to the hardware layer.

The last and lowest layer is the Linux kernel, which forms the basis of the operating system. Basic operations such as power management and file system take place here.

1.3. Android Package File (APK) and Features of APKs

Android package files (APKs) are application files running on android. These files are compressed. As we can see in Figure 1.2, an uncompressed APK file includes res, lib, assets, androidmanifest.xml, META-INF, classes.dex and resources.arsc files, which are used when analyzing an APK.

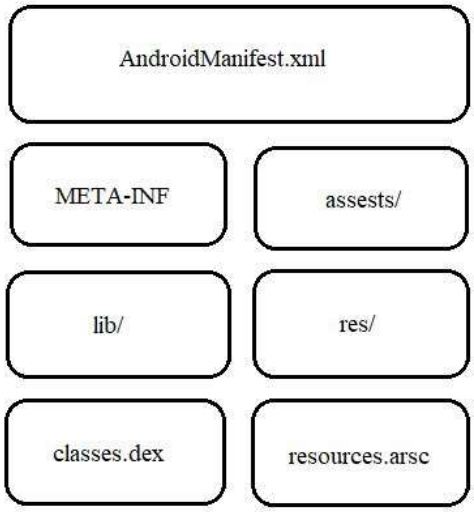

Figure 1.2. Structure of APK

Android package files (APKs) are application files running on android. These files are compressed. As we can see in Figure 1.2, an uncompressed APK file includes res, lib, assets, androidmanifest.xml, META-INF, classes.dex and resources.arsc files, which are used when analyzing an APK.

The classes.dex file contains the compiled code since it is a compiled file, it can be run by Dalvik Machine. Java bytecode is in .dex format. The bytecode needs to be translated into the high-level language to make it easier for people to understand. This file is translated into java using reverse engineering methods. In this way, the codes are examined. Features such as classes and functions are extracted from the codes. It can also compare with code that was previously known to be malicious. Malware detection can be performed by using such methods.

Res files contain resources that cannot be compiled. These resources usually contain XML files. The res file can contain drawable, mipmaps, and layouts. Also, for example, when an external photo is used in the application, it can be stored under this file regardless of the code. The Resources.arsc file is a compiled source file. Examples of these are binary XML files, such as style color and arrays.

The META-INF file contains MANIFEST.MF, CERT.RSA and CERT.SF files. These files provide information about the application, contain the application's certificate, and contain a

list of resources, respectively. Each author must sign his or her application. Since the user's certificate does not match the author's certificate, this process prevents the application from re-signing by another user. This is a security measure taken for android applications. The META-INF file provides metadata about the application, it can be used in malware detection.

Lib contains a platform-independent compiled code. This code is usually C/C++ code. The aim is to enrich the application by using the library of other languages. These libraries are used by the Native Development Kit (NDK).

Assets are files that are excluding android resources, such as game data files about game levels. These files can be read by using AssetManager.

Androidmanifest.xml is the summary file for the application. It has a binary Extensible Markup Language (XML) format. It contains a lot of information about the application, such as package name, permissions, version, etc. Applications use components such as content provider, activity, service, broadcast receiver, and intent. These components are also found in the Android manifest.xml file. If these components are mentioned briefly, the content provider is responsible for processing the data. All screens with interface in the application are considered activity. Services run in the background and do long-term operations and have no interfaces. Broadcast Receiver notifies the relevant locations when an event occurs. Examples of broadcasts in the operating system include low charge and switching the phone to airplane mode. Intent connects components. To share the information in an email application with WhatsApp application, the intent will allow two applications to communicate with each other.

Various features can be extracted from these files statically or dynamically. System-calls, API calls, and permissions, version number, opcodes from class.dex file, last update time, code metrics, broadcast receiver, native code, Control Flow Graph (CFG), activities, application category, meta-data available, bytecode fragments, OS and android framework commands are examples of statically extracted features. Examples of dynamically extracted features include network traffic, number of active communications, power consumption, battery usage, battery temperature, virtual memory features, process information, IP address, CPU, SMS and memory usage, time between sequential system calls and system calls (Rodríguez-Mota, Escamilla-Ambrosio, & Salinas-Rosales, 2017).

When Android first came out in 2008, it wasn't used by too many people, except researchers, academics and malicious people. They were interested in installing malicious software on the system. In 2008, a group of security researchers announced the first attack on android. This attack has 4 different modules. These are: rejecting all incoming calls, turning off the radio, ending all calls and collecting sensitive information and sending it to the attacker. The purpose of this group was not to damage the system but to show the vulnerabilities of the system and encourage it to be safe. Afterward, various malware that wanted to harm the system showed up, such as Android OS, Mobile Spy, FakePlayer, SMS Replicator, zone, Droid KungFu. The general purpose of all of these is to damage the system in certain ways, access sensitive information about the user and use it for malicious purposes. Therefore it is essential to detect Android malware. As mentioned previously, the features that are extracted from the files can be used in malware detection (Castillo, 2011).

1.4. Reverse Engineering

Reverse engineering is a method for analyzing the system. Its main purpose is to understand how the system works and what it does and to find solutions for the system's shortcomings. It can be used in all areas where the system exists.

In the field of software, reverse engineering is based on examining the program. The code is written in the background in each software program. This code can be examined using reverse engineering. Android apps also contain code in itself. It is possible to examine these applications by using reverse engineering. As a result of this examination, various information is obtained. This information is used in various fields. These fields are:

- Reading and understanding code fragments
- Adding new codes to old one’s
- Finding vulnerabilities of the code
- Analyze the malware inside the code
- Reaching sensitive data in the code

There are two types of reverse engineering. One of them is disassembly, the other is decompilation. Disassembly translates the language of a code from machine to assembly. Assembly language is more understandable than machine language for people. Thus, people can easily understand the code. Decompilation translates code from an executable format to a

source code format. The code obtained by reverse engineering cannot be the same as the code written by the developer, but the code is closest to that code.

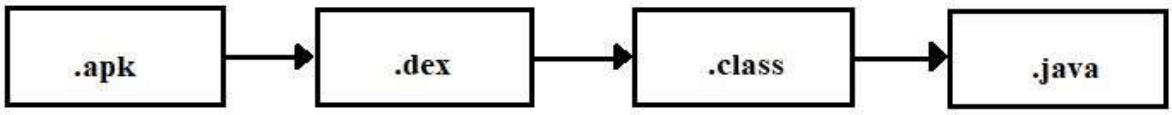

Figure 1.3. Reverse Engineering Phases of APK files

Figure 1.3 shows the reverse engineering steps for an APK file. An APK file is first converted to a .dex file, then to a .class and .java file. The closest code to the code written by the developer is .java file.

1.5. Analysis Tools

There are tools to easily analyze APK files. Information about APK files can be extracted by using these tools. In this section, some of these tools will be mentioned.

Apktool is a reverse engineering tool that returns the resources of the APK file almost to its original state. These resources are resources.arsc, classes.dex and XMLs. Thanks to apktool, the source code can be accessed and converted back into APK file format again.

Dex2jar converts the compiled android code into java source code. The compiled code is in .dex format. Classes can be obtained by converting this to java code. The actual code written by the developer is not obtained. It can convert Dex to Jar and vice versa.

Ghidra is a disassembler for reverse engineering and malware analysis. This tool is open-source and free.

TaintDroid is a dynamic analysis tool. It monitors the behavior of the application and gets the taint parts for investigating and reporting.

Smalisca is a static analysis tool. It is used for smali files and extracts features from them. These features are class information, methods, and properties.

Frida is a toolkit that changing the behavior of the application dynamically. Appmon is an application written using Frida. It records operations of an application, while the application is running. The objection is another application written using Frida. It is used to perform security analysis in running applications.

1.6. Malware Detection Techniques

Malware detection techniques generally divided into 3 parts. They are given in Figure 1.4. These are Signature-based, Behavior-based and Heuristic-based detection. Also, all techniques in itself are divided into 3 categories: static, dynamic and hybrid. These categories are only described under behavior-based detection but all techniques have these categories.

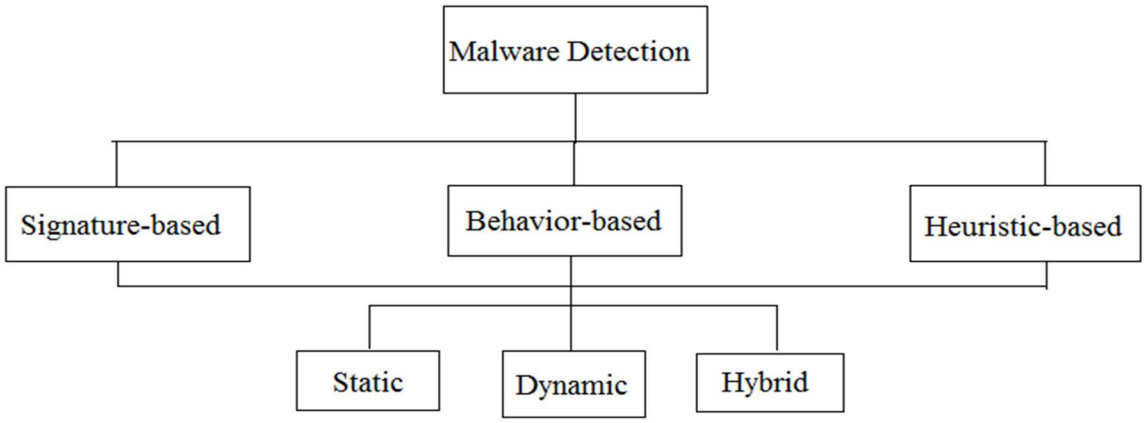

Figure 1.4. Malware Detection Techniques

1.6.1. Signature-Based detection

It (Mathur & Hiranwal, 2013) detects malware with a pattern matching method. A signature is created according to extracted features from the codes, and this signature is compared to those already registered in the database. If pairing is provided, malware is detected. The disadvantage is that if the code is not already stored in the database, malware can not be detected. Another disadvantage of signature-based detection is it needs money and time while extracting a specific signature. The advantage is that if the signature is registered, it can be immediately matched and the misdetection (false positive) rate will be less.

1.6.2. Behavior-Based detection

It (Mathur & Hiranwal, 2013) examines what the program does. It collects programs with the same behavior in one category. So behavior-based detection can detect malware that has similar attributes. While signature-based is specific to each program, behavior-based can

detect various malware types that are similar to each other. Some of these are unknown and polymorphic malware. Since it scans a large amount of behavior, the false positive rate can be huge and detection can take a long time.

1.6.2.1. Static analysis

Analysis without code execution is called static analysis. (Gadhiya & Bhavsar, 2013) The advantage of this analysis is that it does not run the code, so it avoids possible losses beforehand. The disadvantage is that it cannot detect unknown malware.

In the thesis, static analysis is used because it is an easier and faster analysis method according to other analysis methods.

1.6.2.2. Dynamic analysis

After the code is executed, it is detected as malware or not according to the behavior of the program in dynamic analysis. (Gadhiya & Bhavsar, 2013) These codes are usually run in the virtual environment to avoid damaging the computer and their behavior is monitored.

1.6.2.3. Hybrid analysis

The hybrid analysis involves the use of static and dynamic methods together. First of all, the signature is examined as it is done in the signature-based method and then the program is executed and the behavior is observed. In this way, it has advantages in both techniques.

1.6.3. Heuristic-Based analysis

Heuristic-based detection (Bazrafshan, Hashemi, Fard, & Hamzeh, 2013) uses machine learning and data mining techniques to learn the behavior of an executable file. Several features are used for detecting malware. These are API (application programming interface) calls, CFG (Control Flow Graph), N-gram and OpCode.

- API calls are used by most of the programs to request something from the operating system.

- The OpCode is short of operational code. This contains machine language instructions for detecting malware.
- N-gram divides strings into substrings and every substring has a length of N. For example, “malware” with N equal to 3 are substrings of “malware” that are “mal”,”alw”,”lwa”,”war” and “are”.
- CFG represents the control flow of the program as a graph. The graph contains nodes and edges. In CFG, nodes represent statements and edges represent control flow between the statements.

1.7. Machine Learning

Machine learning is the machine's self-learning without human assistance. The machine has no information at first. Based on the examples, it makes various inferences. So, it learns from experience. Machine learning has 3 kinds of learning methods. These are supervised learning, unsupervised learning and reinforcement learning.

In supervised learning, the data is delivered to the machine in a labeled form. And the machine tries to estimate the label of the incoming data by looking at these labeled data.

In the unsupervised method, the data is not labeled. But similar data can be collected in a group. Thus, it is possible to find out which data group is close to the incoming data and make an inference about it.

In Reinforcement learning, the machine does not know the environment. It acts by trial and error method and calculates error. It tries to learn by minimizing the error or gets a reward as feedback and tries to learn by maximizing it. It should receive positive or negative feedback from the system to find the best behavior as a result of its actions.

1.8. The Aim of This Thesis

The aim of the thesis is to detect malicious software installed on Android with high accuracy and minimal margin of error. In this thesis, as a different method, features are statically extracted from APK files. The Fuzzy Set-Based (FSB) weighting method was applied to extracted features. Both malware and benign applications may have the same features. The FSB weighting method will assign a weight to the feature not found in the APK. Thus, a

feature that is actually associated with the APK will not be overlooked. In addition, feature selection has been made using the Firefly Optimization Algorithm (FOA) to highlight more important features.

The remaining chapters continue as follows: Chapter 2 describes how malware is created, how it is detected, and related work about malware detection. In Chapter 3, datasets and methods used will be explained. In Chapter 4, experiments and results obtained will be mentioned. In the last chapter, the topic will be concluded and information about future studies will be given.

2. LITERATURE REVIEW

In this section of the thesis, related work including static analysis, dynamic analysis, hybrid analysis techniques for malware detection, nature-inspired based feature selection methods, Fuzzy Set-Based methods, and Firefly Optimization algorithm are presented.

2.1. Related Work

2.1.1. Static analysis

Santos et al. (2009) used N-gram and k-NN to detect malware. The study aimed to detect unknown malware that cannot be detected with the signature-based method. They chose 3 values. They chose the "N" value to decide the length of the N-gram, k value to determine how many neighbors there would be and d value to determine how hard they would be when detecting malware. For N, the most appropriate length was determined as 4, 7 for k and 2 for d. With these results, 91.25% accuracy was reached. When they wanted to reduce the false positive rate to zero, they chose n as 4, k and d values were 17, 74.37% detection rate is obtained.

Chen et al. (2015) tried to detect malware using clone detection. Benign apps were collected from AppChina and malware apps were collected from the Android Malware Genome Project. Reverse engineering was applied for getting source code. For reverse engineering, the dex file was converted to jar file and source code was getting with using jar decompiler. NiCad was used for clone detection. They used three types for threshold. Type-1 and Type-2 were 100% similarity matching threshold. Type-3 was a 70% similarity matching threshold. When type-1 was an exact clone, type-2 was renamed clone. As a result, they detected malware with 96.88% accuracy in type-2 malware.

Most of the studies used signature-based and n-gram techniques for malware detection, on the contrary, Anderson et al. (2011) used graph-based methods for showing superiority according to other methods in order to track run time execution. Their dataset had 1615 malware and 615 benign samples. After constructing the similarity matrix between graphs, this similarity matrix was classified with SVM. Graph similarity calculated by using three different kernels. These were Gaussian, Spectral and a combination of them. Also, they classified with n-grams

and the five most popular anti-virus programs. As a result, they demonstrated that their combined kernel methods reached the best result with 96.41% accuracy.

De La Rosa et al. (2018) used three kinds of features using static analysis. These were bytes, basic and assembly features. Byte features were easy to extract and assembly features were required disassembling the byte code so it was difficult to compute. In their study, they used these features and deep learning methods for classification. In their meta-model, they compared the results of bytes, basic and assembly after that they decided which model is more convenient (time, accuracy, etc.) for classification with using their meta-model deep learning. Accuracies of Bytes (B), Bytes Basic (BB) and Bytes Basic Assembly (BBA) were 86.36%, 90.07% and 90.42, respectively. Their average times were 0.06, 4.12 and 60 seconds, respectively. In the meta-model, accuracy was 90.42% and the average time was 7.23. As a result, the meta-model was very close to BB in the timing and it was higher than BBA in the accuracy.

Obeis and Bhaya (2018) used API calls and the shingling method for detection malware. They used two different datasets. These were CSDMC2010 and APIMDS. For the CSDMC2010 dataset, they used 70% as training and 30% as testing. For the APIMDS dataset, they used all datasets as testing. They chose different shingle length but the best result for k-shingle is 2. As a result, In the CSDMC2010 dataset, it was reached 98.8% accuracy and in the APIMDS dataset, it was reached 99% accuracy.

Saxe and Berlin (2015) extracted 4 different features from malware and benign samples. These features were contextual byte features, PE import features, String 2d histogram features, and PE metadata features. For classification, they used a deep neural network. Also, Bayesian calibration was calculated for samples classified as malware. It decided that these samples were really malware or not with giving a probability. As a result, using four different features together, they reached a 95.2% true positive rate and a 0.1% false-positive rate.

2.1.2. Dynamic analysis

Firdausi et al. (2010) detected malware by using dynamic behavioral detection. They used portable executable files for benign and malware dataset. They used Anubis that is a free dynamic analysis service on the internet and the report of Anubis is the XML file format. XML files were installed, parsed and important values were selected inside them. Two

weighting methods were used for all attribute values. These were binary weight and term frequency weight. After that KNN, SVM, Naïve Bayes and J48 classification methods were applied. They also applied a feature selection method. As a result, J48 had the highest accuracy of 96.8% with term frequency weighting and no feature selection.

Çarkacı and Soğukpınar (2016) tried to detect metamorphic malware using MAIL (Malware Analysis Intermediate Language). Firstly, they converted binary code to OpCode that had 26 MAIL instructions inside it. After that, TF, IDF and TF-IDF methods were used for extracting features. Codes that converting to MAIL was divided into 80% training and 20% testing. Random Forest, KNN, SVM, Logistic algorithms were applied for classification. Also feature selection methods were applied. As a result, 2 valuable features were selected and the performance of classifier increased from 93% to 100%. TF was the best feature extraction method. SVM and KNN were the best classifiers.

Ni et al. (2016) tried to detect malware in real-time. For this purpose, they used android apps. Benign apps collected from the Goggle play store and Anruan Market. Malware apps collected from the Android Malware Genome Project. API calls, permission uses, and run-time behavioral features were collected with the automated testing platform. They also used sensitive behavior sequences. In total, they extracted 24 malicious behavior sequence patterns. For every app, they looked partial sequence pattern matching. This partial sequence pattern was N and N was changed between 2 and 5. According to these features, they classified with SVM and Naïve Bayes. As a result, when N was equal to 5, the SVM classification result reached 99.0% accuracy.

2.1.3. Hybrid analysis

Pektaş et al. (2018) detected windows based-malware with using dynamic analysis and API calls for features. Firstly, they collected dynamic analysis results from .exe files. After that CM–SPAM algorithm was used for sequential pattern mining. For each pattern, they assigned a weight with TF-IDF calculation. Each sample represented as a vector using pattern calculation. Finally, they classified with three machine learning methods. These were Random Forest, KNN and SVM. As a result, Random Forest was the best with 99% F-measure.

Yuan et al. (2014) used three different features. These were required permission, sensitive API and dynamic behavior. Static analysis was used for required permission and sensitive

API features. Dynamic analysis was used for dynamic behavior. The contagio mobile was used for malware samples and the Google Play Store was used for the benign samples. Deep learning was used for classification. Different numbers of neurons and layers were used. The best result was 96.5% accuracy with 3 number of layers and 150 neurons in each layer. Also, MLP, SVM, Naive Bayes, LR and C4.5 classifiers were used and their accuracies were 79.5%, 80.0%, 79.0%, 78.0% and 77.5%, respectively. As a result, the best classifier was a deep neural network.

2.1.4. Nature Inspired algorithms

Nature Inspired Algorithms are algorithms that optimize problems inspired by natural phenomena, natural laws and living things in nature. The Intelligent Water Drops Algorithm is an example of natural phenomena, the Central Force Optimization Algorithm is an example of natural laws, and the Firefly Optimization Algorithm is an example of living things in nature.

Adebayo and AbdulAziz (2014) used static analysis. APK files were used from "contagion dump" for malware samples and "google play store" for benign samples. Important features were selected by using the Apriori Algorithm (AA) and Particle Swarm Optimization (PSO) from APK files. After that samples were classified with classification methods. The study aimed to increase the rate of malware detection using supervised and unsupervised learning together. As a result, 3 different categories were examined using only AA, using only PSO and using both PSO and AA. Classifiers were Neural Network, Classification-based Multiple Association Rule (CMAR) and Bayesian classifier (BC). The best result for classifiers was BC and for feature, selection using both PSO and AA. 97.2% accuracy value which is the best result of this study, is obtained with the Bayesian classifier and the hybrid of PSO and AA methods feature selector.

When detecting Android malware, some permissions can be seen in both the benign and the malware classes. Razak et al. (2018) used the Bio-Inspired Algorithm to eliminate these permissions. Malware samples were obtained from the Drebin dataset and benign samples were obtained from the Google Play Store using Androzoo. There are a total of 8500 samples including 3500 malware and 5000 benign samples. Particle Swarm Optimization (PSO), Evolutionary Computation and Information Gain (IG) are used for feature selection. Two of

these are the Bio-Inspired Algorithm. If the feature is found in the APK, the weight is 1, if not the weight is 0. RF, MLP, KNN, Adaboost and J48 classifiers are used. Ten-fold cross-validation is applied. 95.6% true positive rate is obtained using PSO and Adaboost.

It is difficult to deal with a large dataset. Therefore, Jeyarani and Pethalakshmi (2016) perform feature selection and classification using 4 different large datasets, WILT, ORL, LC, and CTG. For this, the Fuzzy Entropy and FOA were used. Fuzzy Entropy is an extended version of Shannon Entropy. And it uses the Fuzzy C-means Clustering Algorithm in its calculation. For each feature, Fuzzy Entropy is calculated, and those greater than the predetermined threshold are selected as the important features. Then the classification is made by using the FOA. The best classification result is selected. The SVM classifier is used for classification. The proposed method is compared with the Fast Correlation-based Filter (FCBF) method. In 4 datasets, the Optimized Feature Selection using Fuzzy Entropy and Firefly (OFEFLY) algorithm outperformed FCBF. WILT, ORL, LC, and CTG dataset are reached 86.86%, 100%, 87.50% and 85.14% accuracy, respectively in OFEFLY. WILT, ORL, LC and CTG dataset are reached 74.50%, 91.55%, 79.42% and 79.28% accuracy, respectively in FCBF method.

Anbu and Anandha Mala (2019) used the KC1 dataset, which is written in C ++ and has a defect of 15.5%, to detect defects in the software. They proposed a Feature Selection (FS) algorithm based on Firefly Optimization. They used SVM, NB and KNN classifiers. SVM with FS has achieved 2.8% better accuracy than KNN with FS. As a result, SVM and firefly feature selection outperformed other methods.

Al-Sheshtawi (2010) is used the Clonal Selection Algorithm with 8 different types: AIRS1, AIRS2, AIRS2 Parallel, CLONALG, CSCA, IMMUNOS-1, IMMUNOS -81, and IMMUNOS -99 as a classifier. They used Portable Executable (PE) files. They obtained 100 worms from VX Heavens virus collections and 100 benign executables from Windows XP and application installers. They extracted the features using n-gram. They took the number n as 4. They ranked the features using information gain according to their importance. If the feature is found in executable file the weight is 1, if not the weight is 0. Tenfold cross-validation is applied. Consequently, As a result, AIRS2 and AIRS2-Parallel accuracies were the best with 95.6522%.

Various Nature Inspired Algorithms have been used in malware datasets for classification and feature selection. However, according to the information we have obtained so far, the FOA for Android malware has not been used as a feature selection method. On the other hand, in this study, this optimization algorithm is used and demonstrated its effectiveness with detailed experiments.

2.1.5. Fuzzy Set-Based methods for malware detection

Fuzzy sets creates abstract and flexible structures. It is used to smooth sharpness in data. In this way, the data can be adapted to the system more meaningfully and more accurately.

Alther & Barukap (2017) were extracted permissions from the AndroidManifest.xml file. The extracted features were ranked with Information Gain and 24 important permissions were selected. Then, Fuzzy C-means Clustering (FCM) algorithm was used to determine the center of clusters and the number of clusters. This algorithm increased the learning process and the accuracy of the adaptive neuro-fuzzy inference system (ANFIS). ANFIS combines neural networks with fuzzy logic. For ANFIS, Gaussian (gauss), sigmoid, generalized bell (gbell), and trapezoidal membership function (trap) have implemented as 4 different membership functions. The lowest Root Mean Squared Error (RMSE) was the sigmoid membership function. For this reason, the sigmoid membership function is used. The dataset of GNOME project is used. Ten-fold cross-validation is applied. FCM-ANFIS, k-ANFIS, ANFIS, and DENFIS classifiers achieved 91%, 75%, 70%, and 70% accuracy rates, respectively. Thus, the proposed FCM-ANFIS method achieved the best results.

Puram and Singh (2018) were received reviews from the App-Review-Dataset for 80 applications. These reviews were used to determine whether the app is a fraud. There are many ways to determine if an app is a fraud, but these methods usually distinguish between good and bad. In other words, they are examined in two categories. However, in their study, they found that reviews with more than 2 categories would give more accurate results for the fraud app. To do this, they divided the reviews into different groups using Fuzzy Logic and applied different threshold values. The best result was achieved with 93.75% accuracy when there were 5 groups and the threshold value was 0.2.

To the best of our knowledge, in the literature, there are several fuzzy-based studies for malware detection problems, but there are few studies used as the Fuzzy Set-Based (FSB)

weighting approach. These studies were implemented on SMS spam messages and Cyber Terror and Extremism (CTE) dataset. The superiority of the fuzzy weighting approach has been demonstrated experimentally (Uzel & Saraç Eşsiz, 2019; Uzel, Saraç Eşsiz, & Özel, 2018). In this study, the effectiveness of this method is observed for malware detection problem.

3. MATERIALS AND METHODS

This section includes details about the material used in this study and the proposed method.

3.1. Dataset

Android Malware Dataset (AMD) (Wei, Li, Roy, Ou, & Zhou, 2017), CICAndMal2017 (Lashkari, A. Kadir, Laya, & Ghorbani, 2018) dataset and contagio mobile malware samples (Parkour, contagio mobile, n.d.) are used together in this thesis. All benign samples are from the CICAndMal2017 dataset. There are a total of 220 benign and 224 malware samples in the new dataset. There are 5 types of malware inside the dataset such as Adware, Ransomware, Backdoor, Trojan, and SMS malware. All of them have types inside it. Adware includes airpush, andup, dowgin, edwing, feiwo, gooligan, kemoge, koodous, kuguo, kyview, minimob, mobidash, shuanet, utchi, and youmi types. Ransomware includes aples, charger, fusob, jisut, koler, lockerpin, pletor, porndroid, ransombo, roop, simplelocker, svpeng, and wannalocker types. Backdoor includes androrat, droidkungfu, fakeangry, fjcon, fobus, gingermaster, goldream, obad, spambot, triada and univert types. Trojan includes fakeav, fakedoc, faketimer, fakeupdates, ksapp, lnk, mmarketpay, mobiletx, mseg, mtk, nandrobox, and uptdkiller types. Smsmalware includes beanbot, biige, boxer, cova, erop, fakeinst, fakemart, fakenotify, fakeplayer, gumen, jikafe, leech, mazarbot, ogel, opfake, plankton, rums, smskey, smssniffer, spybubble, stealer, tesbo, vidro and zsone types.

3.2. Data Preprocessing and Feature Extraction

Smali files and androidmanfiest.xml files were extracted from APK files by using apktool.
The code was written in python to extract features from these extracted files.

API call candidates such as "Landroid/", "Lcom/android/internal/util", "Ldalvik/", "Ljava/", "Ljavax/", "Lorg/apache/", "Lorg/json/", "Lorg/w3c/dom/", "Lorg/xml/sax", "Lorg/xmlpull/v1/", "Ljunit/" were extracted from smali files (Android Developers, n.d.). Invoke-virtual, invoke-direct, invoke-static, invoke-interface and invoke-super lines are important for smali files. In Figure 3.1, Landroid/content/SharedPreferences is extracted as a feature from the smali file.

```
move-result-object v0

.line 247
.local v0, "settings":Landroid/content/SharedPreferences;
const-string v1, "WAS_PROGRESS_DONE"

invoke-interface {v0, v1, v2}, Landroid/content/SharedPreferences;->getBoolean(Ljava/lang/String;Z)Z

move-result v1

iput-boolean v1, p0, Lcom/software/installer/Main;->wasProgressDone:Z
```

Figure 3.1. Sample Smali File

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<application android:debuggable="true" android:icon="@drawable/icon" android:label="@string/app_name">
    <receiver android:name=".Notifier">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED"/>
            <category android:name="android.intent.category.HOME"/>
        </intent-filter>
    </receiver>
```

Figure 3.2. Sample AndroidManifest.xml File

Categories, actions, and permissions were extracted from Andoridmanifest.xml. For example, in Figure 3.2, android.intent.action.BOOT_COMPLETED is extacted as an action, android.intent.category.HOME is extracted as a category and android.permission.SEND_SMS is extracted as a permission.

```
android.intent.action.BOOT_COMPLETED
android.intent.action.MAIN
android.intent.category.HOME
android.intent.category.LAUNCHER
android.permission.INTERNET
android.permission.READ_PHONE_STATE
android.permission.SEND_SMS
android.permission.WRITE_EXTERNAL_STORAGE
Landroid/app/Activity
Ljava/io/BufferedReader
Landroid/os/AsyncTask
Landroid/telephony/SmsManager
Landroid/telephony/TelephonyManager
```

Figure 3.3. Feature Document for an APK file

Then, they were merged into one file. As a result, each APK file has a feature document. Figure 3.3 shows example feature document for an APK file. Also, they are uniquely collected in a single file. In total 8308 features are extracted from APK files.

3.3. Classification Methods

In this thesis, Radial Basis Function Network, Naïve Bayes, Random Forest, Random Tree, and K-Nearest Neighbor classifiers were used for classification.

3.3.1. Radial Basis Function Network classifier

Radial Basis Function Network (RBFN) is a kind of feedforward neural network. In a feedforward neural network, it has three layers. These layers are the input layer, hidden layer, and output layer. RBFN differs from a feedforward neural network with transfer function. Radial Basis Function (RBF) is a transfer function of RBFN. This is a non-linear function. The Gaussian function is generally used for RBF.

$$o_{di} = \exp\left(\frac{-\|Y_d - C_i\|^2}{2{\sigma_i}^2}\right) \quad i = 1,2,3 \dots K \tag{1}$$

In equation 1, Y_d is the input vector and o_{di} is the output of the i^{th} hidden layer. C_i is the center of the i^{th} hidden layer. For finding centers of hidden neurons, the K-means clustering algorithm is used. σ_i is the width of the i^{th} hidden layer. Width is an acceptable distance for each hidden neuron. According to these values, it calculates an error of the prediction then determines the class of the data (Broomhead & Lowe, 1988).

3.3.2. Naïve Bayes classifier

Bayesian classifiers are statistical classifiers. Class prediction of an instance made with a probability value. The formula of this probability value is given in equation 2.

$$Prob(cla, X_1, \dots, X_m) = Prob(cla) * \prod_{i=1}^{m} Prob(x_i \mid cla) \quad (2)$$

Cla is the class label and *m* is the number of attributes. x_i is the value of an attribute X_i. The probability of i^{th} attribute is calculated with multiplying class probability and the conditional probability of *ith* attribute (Maron, 1961).

3.3.3. Random Forest classifier

Random forest is a kind of decision tree classifier. It is both used for classification and regression. Decision tree, depending on the rules, incorporates features into the classification by simulating the tree. It continues to add features until the last feature that contributes to the classification. In a random forest, unlike the decision tree, features are not added to trees according to a specific rule. In other words, they are added randomly. Multiple trees are created. These trees are finally combined and the best result is obtained. Creating trees in different shapes also eliminates the overfitting problem. It is a fast algorithm. As the number of random trees increases, the accuracy rate will increase but the speed will decrease (Ho, 1995).

3.3.4. Random Tree classifier

The random tree is a kind of Decision Tree classifier. It is a supervised classifier. It looks like a Random Forest but it has a small different from the Random Forest.

The Random Tree consists of many tree communities. Each tree classifies the input vector. Whichever class receives more votes, the input data is classified as belonging to that class. The Random Tree is a mixture of Random Forest and Decision Tree (Breiman, 2001).

3.3.5. K-Nearest Neighbor classifier

The K-nearest neighbor is a well-known classifier due to its easy implementation and high accuracy. It uses a distance metric for classifying a new instance. For this purpose, it

calculates distances between the new instance and k training instances. The value of k can change according to how many neighbors are desired. The class labels of neighboring instances are checked, and whichever class label is the majority, is determined as the label of the new instance. Euclidian Distance is generally used as a distance function (Aha & Kibler, 1991).

$$E_d = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2} \tag{3}$$

In equation 3, E_d represent Euclidian distance, *x* and *y* represent instances, *i* and *k* represent *ith* attribute and the total number of attributes, respectively.

3.4. Weighting Methods

In this section, the weighting methods that are used in this study are explained.

3.4.1. TF-IDF weighting method

It is a known and highly accurate weighting method. There are two different calculation operations. One of these is to calculate the term frequency (TF) in the document. The other is the inverse document frequency (IDF).

$$TF(t) = \frac{\#\ of\ times\ term\ t\ in\ document\ D}{total\ occurrences\ of\ all\ terms\ in\ document\ D} \tag{4}$$

$$IDF(t) = \log(\frac{Total\ number\ of\ documents}{\#\ of\ documents\ with\ term\ t\ in\ it}) \tag{5}$$

The calculation of the term frequency is as shown in equation 4. This calculation counts the number of times a term is used in that document. The calculation of the IDF is as shown in equation 5. The purpose of the IDF calculation is to calculate how valuable that term is among all documents. According to equation 5, it can be said that frequently seen terms have less importance. By multiplying these two values, the weight of a term is calculated for the document (Salton & Buckley, 1988).

Table 3.1. Sample documents for weighting calculation

Document Number	Terms in the document
1	Landroid/telephony/SmsMessage, Ljava/lang/String, android.permission.NFC
2	Landroid/telephony/SmsMessage, Ljava/lang/String , Ljava/lang/Error
3	Landroid/telephony/SmsMessage, android.permission.NFC

Sample documents are given in Table 3.1 for a better understanding of the computational logic of TF-IDF and FSB, the calculation will be done through sample documents.

Table 3.2. TF-IDF term weights for sample documents

Document Number	Document vectors (term weights)			
	Landroid/telephony /SmsMessage	Ljava/lang /String	android.permission. NFC	Ljava/lang/ Error
1	0	0.058	0.058	0
2	0	0.058	0	0.157
3	0	0	0.088	0

In Table 3.2, weights of all terms are calculated using TF-IDF weighting for each sample document.

3.4.2. Fuzzy Set-Based weighting method

It has not been used as a weighting method in previous studies for malware detection. From this point of view, it is a different and new method. The purpose of this weighting, unlike other known weighting methods, assigns a weight to the term, even if the term is not found in the document. In this way, the relevance of the term with the document is determined. To do this, it uses the Jaccard index (JI) and a different weighting calculation method, which is given in equation 6.

$$c(t,u) = \frac{\#\ of\ documents\ cantaining\ both\ terms\ t\ and\ u}{\#\ of\ documents\ containing\ at\ least\ one\ of\ terms\ t\ and\ u} \quad (6)$$

Table 3.3. JIs of all terms for FSB calculation

c(t,u)	Landroid/telephony/ SmsMessage	Ljava/lang/ String	android.permission .NFC	Ljava/lang /Error
Landroid/telephony /SmsMessage	1	0.67	0.67	0.33
Ljava/lang/String		1	0.33	0.5
android.permission. NFC			1	0
Ljava/lang/Error				1

As seen in Equation 6, the purpose of the JI is to calculate the similarity between terms. To do this, it uses the number of documents in which two terms exist simultaneously in a document and the number of documents in which at least one of the two terms exists. The ratio of these two values gives the JI. In Table 3.3, JI of the terms are calculated for sample documents (Zadeh, 1965).

$$W(D,t) = 1 - \prod\nolimits_{u \in D} (1 - c(t,u)) \quad (7)$$

Table 3.4. FSB weighting calculation for sample documents

W(D,t)	Landroid/telephony/ SmsMessage	Ljava/lang/ String	android.permission .NFC	Ljava/lang /Error
Document-1	1	1	1	0.67
Document-2	1	1	0.78	1
Document-3	1	0.78	1	0.33

It performs 2 different assignments to calculate the weight of the term in the document. If the term exists in the document, it assigns the weight 1 directly to that term. If the term does not exist in the document, it calculates the FSB weighting of the term. Therefore, it is learned how much that term belongs to that document.

When calculating the FSB weighting, the JIs of the term not found in the document with the terms found in the document are calculated. OR operation is applied to JIs. As a result of this operation, the relationship between the term and that document is calculated. The calculation is formulated in Equation 7 (Zadeh, 1965). In Table 3.4, FSB weighting was calculated for sample documents. Let us explicitly calculate the weight of Ljava/lang/Error for document 1. For this purpose, it is necessary to calculate the JIs of Ljava/lang/Error with 3 terms in the document. Landroid/telephony/SmsMessage, Ljava/lang/String and android.permission.NFC's JIs with Ljava/lang/Error are 0.33, 0.5 and 0, respectively. For calculation, all JIs are subtracted from 1 and multiplied by each other and finally subtracted from 1 again, the weight of Ljava/lang/Error for the 1st document is calculated. The calculation is as follows: (1 - ((1-0.33) * (1-0.5) * (1 - 0))) = 0.67.

3.5. Feature Selection Methods

In this section, feature selection methods that are used in this study are explained.

3.5.1. Relief-f

Kira and Rendell (1992) proposed the Relief feature selection. This proposed method is used for binary classification. In this method, important features were selected by calculating their dependence on each other. To find the dependency of a feature with classes, it finds the

closest instance in the class to which the feature belongs, and finds the closest instance in the class to which the feature does not belong. Then it gets the difference between the two.

$$W[f] = W[f] - diff(f, R, H)/m + diff(f, R, M)/m \tag{8}$$

As seen in equation 8, for calculating the weight of an f feature, find the nearest hit (H) and the nearest miss (M) to the randomly selected R instance and subtract the differences. “m” is used for normalization. This formula is repeated for all features. After weights were determined for all features, those above a certain threshold were selected as important features. Since relief is only used in binary classification, Kononenko (1994), has developed the Relief-f method for multiple classifications. Here, unlike the other method is to select the closest instance for each class that does not belong to the feature and calculates according to their contributions.

3.5.2. Chi-squared

The Chi-square test (Yates, 1934) tries to find out the relationship between the expected value and the observed value between the two events. In feature selection, to see if a feature is important for the classification, it looks at the status of the feature when it belongs to the class and when it doesn't, and determines whether it is independent of the classification. If the feature is class-dependent, the chi-square test will yield a high result. If the property does not affect the classification, the result of the Chi-square test will be low and that feature will not be selected.

$$X^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i} \tag{9}$$

In equation 9, the number n indicates the number of features. o_i is the observed value e_i is the expected value.

3.5.3. Information Gain

It is an entropy-based feature selection method. Based on the information gained from the feature, it is decided whether the feature is important or not. Information gain is determined by entropy. Entropy refers to the disorder of the system. As entropy increases, uncertainty increases. The less entropy of a system, the greater the information gain.

$$E = -\sum_{i=1}^{n} (\log_2 \frac{nc(i)}{N}) * \frac{nc(i)}{N} \tag{10}$$

Entropy is calculated as in equation 10. In the equation, *n*, *nc(i)* and *N* represent the number of classes, the number of instances for class *i* and the total number of samples, respectively. (Han & Kamber, 2006)

$$E_i = \sum_{k=1}^{n} \frac{nc(k)}{N} * \sum_{m=1}^{nd} -\frac{ncd(k,m)}{nc(k)} (\log_2 \frac{ncd(i,m)}{nc(i)}) \tag{11}$$

$$B(i) = E - E_i \tag{12}$$

In the equation 11, entropy is calculated for feature *i*. *n*, *nc(k)* and *N* are the numbers of different values of the feature, the number of samples belonging to the *k* value of the feature and the total number of samples in the data set, respectively. In the right side of the product, *nd* and *ncd(k,m)* represent the number of classes in the data set and the number of samples representing the class *m* for the feature's *k* value, respectively. In the equation 12, the entropy

value in the equation 11 is subtracted from the entropy value in the equation 10 and the information gain is found for feature *i*. Entropy of all features are calculated in the same way. As a result, important features are selected (Han & Kamber, 2006).

3.5.4. Firefly Optimization Algorithm as a dimension reduction

The FOA is one of the well-known nature-inspired algorithms used for optimization. The name of the algorithm comes from fireflies that are modeled in the construction of the algorithm. At the base of the initiation, fireflies have a single genus (unisex) and are attracted to each other. The attractiveness of a firefly is directly proportional to its brightness. In other words, the brighter firefly will be more attractive. The result is that the less bright firefly is attracted by the brighter one. And the distance plays an important role in brightness and attractiveness. If there is a firefly that is brighter than a firefly, this firefly will move towards the bright one. If there are no more bright fireflies, they move in random directions. The goal is that firefly finds the best place for itself. The result of the algorithm is the firefly, which has the best fitness value.

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2} \tag{13}$$

$$\beta_r = \beta_0 * e^{-\gamma r^2} \tag{14}$$

In Equation 13, the distance calculation formula between the two fireflies is given. x_i and x_j represent firefly *i* and firefly *j*, respectively. Equation-14 shows the attractiveness function of a firefly. β_0 is the initial light value. γ is an absorption coefficient. If γ is infinite, it means that air is foggy. "*r*" is a distance between fireflies (Yang, 2010).

$$x_i = x_i + \beta_0 * e^{-\gamma r^2}(x_j - x_i) + \propto (rand - \frac{1}{2}) \quad (15)$$

In equation 15, i[th] firefly position is updated. In the formula, x_j is the best firefly. The distance between x_i and x_j is calculated according to distance formulas such as Ecludian Distance. β_0 is a light value and $\propto$ is randomness coefficient factor which control the randomness. Using a light value and a randomization parameter, a small random number is added so that it is not only connected to the best. In this way, all fireflies except the best firefly are updated using this formula at each iteration (Yang, 2010).

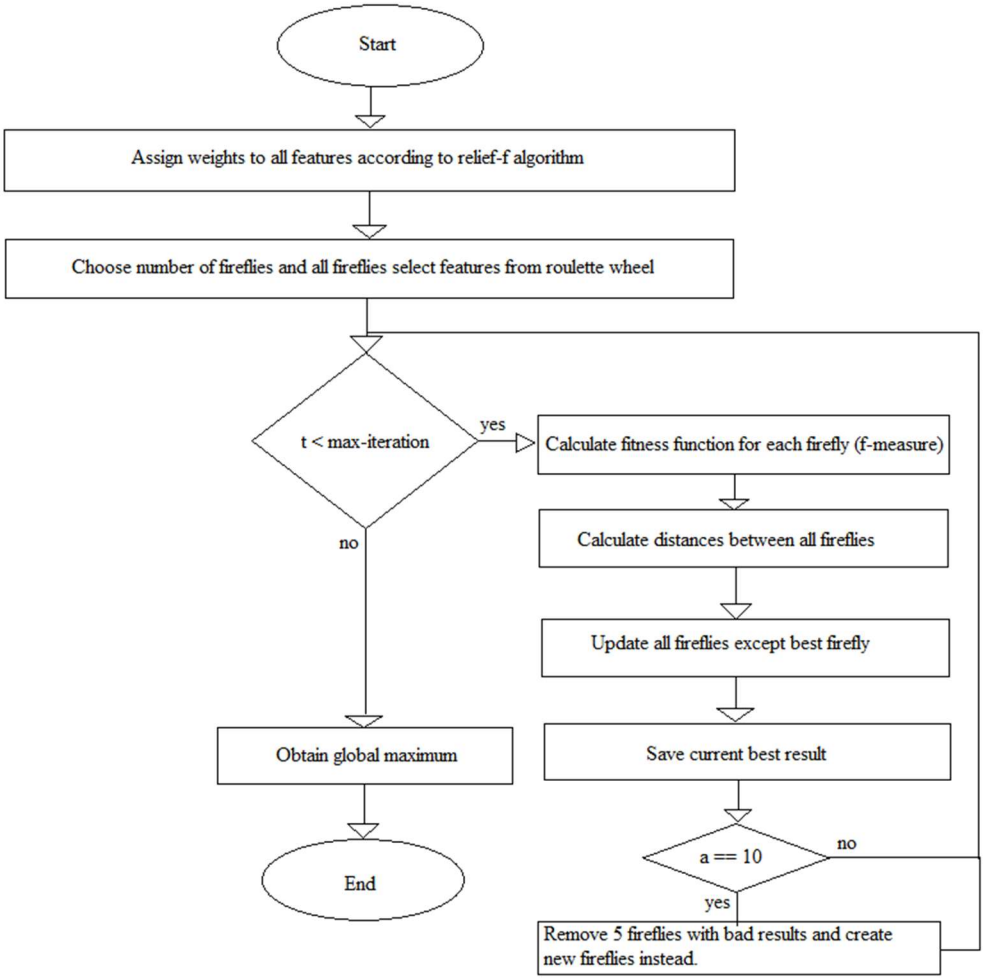

Figure 3.4. The basic steps of the firefly is used in this study

Figure 3.4 describes the basic steps of the firefly algorithm used in the study. First, the Relief-f weight of each feature is calculated. The calculated Relief-f weights are placed on the roulette wheel. In this way, the firefly chooses any number of features from the roulette wheel. 40 fireflies are used for feature selection. According to our previous experiment 40 is the optimum number for firefly population. Each firefly selects different features from the roulette wheel. If predetermined maximum iteration is not reached, the f-measures of each firefly are calculated with RF and KNN classifiers. The distances between all fireflies are calculated. The difference between f-measures is used for the distance value. Firefly, which gives the best results in the current iteration, is stored until a better result is achieved in other iterations. And to ensure global search, every 10 iterations, 5 fireflies with poorest results are removed from the system and new ones are selected from the roulette wheel. These values (10 and 5) were determined empirically. Every iteration, all fireflies except the best firefly are updated. Since fireflies are updated, these values are recalculated at each iteration. In other words, these values are calculated until the maximum iteration is reached. The maximum iteration value is 250 because over 250 iteration there was no diffirent solution. The best results are obtained when max iteration is reached. In this way, the most important features are selected.

Figure 3.5 describes the update phase for fireflies in detail. The update phase for each iteration is calculated for all fireflies except for the current best firefly. With each iteration, the f-measures and distances are updated. And the distances are scaled between 5 and 20, empirically. Fireflies get features from the best firefly in inverse proportion to their distance. Thus, firefly, which is closer to best, is more influenced by the best. For each firefly, the best f-measure firefly is chosen in the neighborhood of half the firefly number. Since the firefly number is used as 40 in this study, the firefly with the best f-measure in 40/2 = 20 neighborhoods is chosen, empirically. In this way, each firefly chooses its best firefly. Each firefly takes features from its own best in the inverse proportion to its distance. To do this, the number of selected features is divided by the scaled distance, and the result tells that how many features to transfer. The best firefly's features are placed on the roulette wheel. The desired number of features is selected from the roulette wheel. Thus, for each firefly, that number of properties are transferred from its best. Once all fireflies except best have been updated, the update ends for one iteration. This continues until the max iteration.

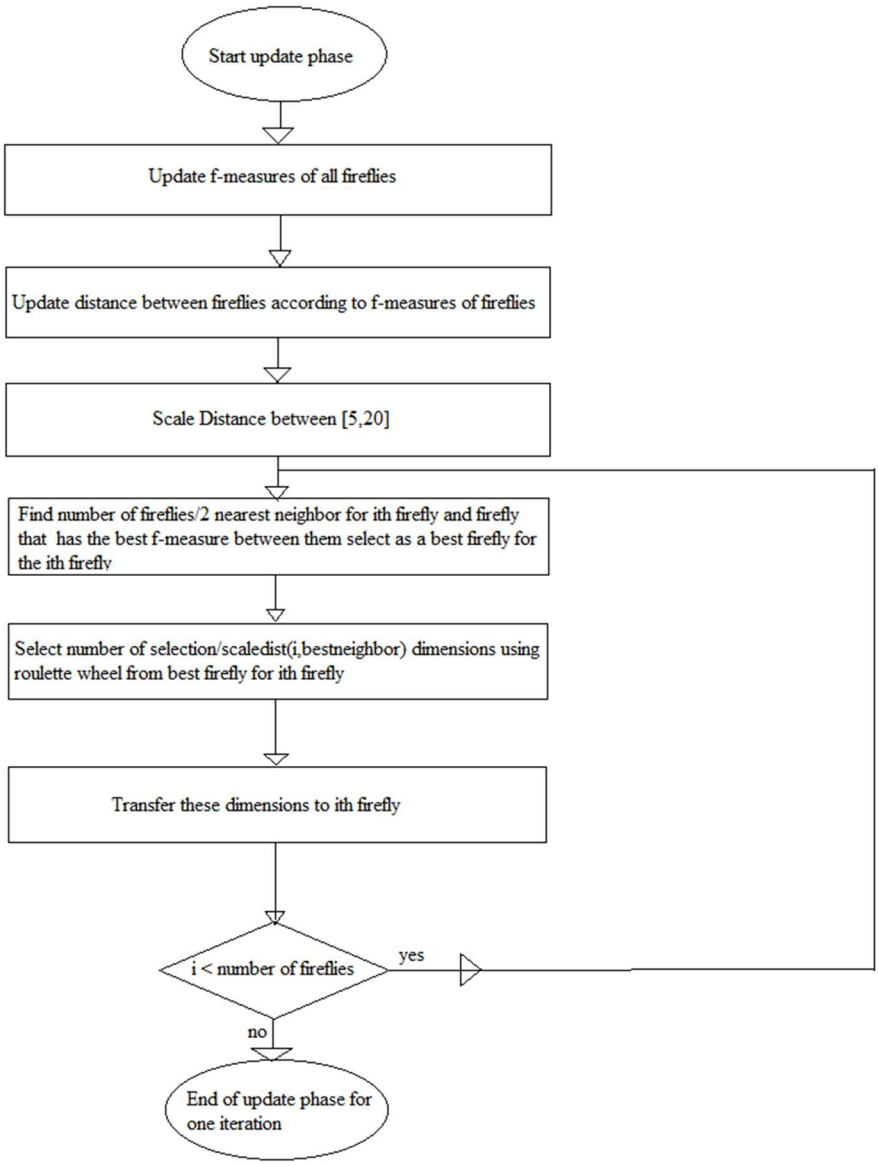

Figure 3.5. The update phase that is used in this study for firefly solution in detail

Selected parameter values have been tested. Good and optimum values were chosen. As a result, the global best firefly's features were used for classification.

Table 3.5. Sample Fireflies

Fireflies	Selected features						Distance with best	f-measure
firefly-best	10	20	30	40	50	60	0	90%
firefly-a	15	35	45	55	65	70	2	88%

Sample values are given in Table 3.5 for a better understanding of the update phase. There are 2 fireflies in the sample system. These fireflies choose 6 features from the roulette wheel. One of them is the best firefly. The features selected for the best firefly are placed on the roulette wheel. For firefly-a 6/2 = 3 features are selected from the roulette wheel. In other words, 3 features are transferred from firefly-best to firefly-a. Let's say 1th, 3th and 5th dimensions are selected from the roulette wheel for the firefly-a. The new values will be as in Table 3.6. Thus, the update is completed for one iteration. In addition, as the number of fireflies increases, fireflies will select features from their best neighborhood fireflies. Moreover, in the first step and update phase, if a firefly has selected a feature more than once, the excess will be deleted and values that are not in the firefly are selected from the roulette wheel. This prevents a firefly from selecting a feature more than once.

Table 3.6. Sample Fireflies after updating

Fireflies	Selected features					
firefly-best	10	20	30	40	50	60
firefly-a	10	35	30	55	50	70

For the other iteration, new f-measure and new distance values are calculated. And the update process takes place again. This cycle continues until the maximum iteration.

3.6. Classification Performance Metric

Classification Performance is measured with Accuracy. Confusion Matrix that is shown in Table 3.7, is used for calculating Accuracy.

Table 3.7. Confusion Matrix

		Predicted Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Accuracy is the percentage of correctly classified instances in the dataset. Accuracy calculation is shown in equation 16.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{16}$$

4. RESULTS AND DISCUSSIONS

The results obtained with TF-IDF and FSB weighting methods before the feature selection are shown in Table 4.1. Tenfold cross-validation is used for all methods. According to the table; RF, RT and RBFN classifiers have better results with FSB weighting, NB and KNN classifiers have better results with the TF-IDF weighting method. As a result, the highest accuracy rate is 92.34% with FSB weighting and the RF classifier. If the classifier evaluates the features as dependent on each other, FSB weighting gives worse results. If the classifier evaluates the features individually, FSB weighting gives better results. This is because FSB weighting brings the data closer to each other, so it's better to separate the data instead of making it closer and more incomprehensible. Since the logic of tree classifiers is to separate the data, it will give good results. The conditional probabilities of features in NB and the neighboring account in KNN are movements to bring the data closer. Therefore, NB and KNN are expected to give worse results than RF.

Table 4.1. Classification results for FSB and TF-IDF weightings with all features

Weighting Methods	NB	KNN	RF	RT	RBFN
FSB	84.91	89.86	**92.34**	**86.26**	**82.43**
TF-IDF	**88.29**	**90.54**	90.99	83.11	79.73

When classifying using FSB weighting, we encountered a problem that we did not encounter in our previous studies with FSB weighting. (Uzel & Saraç Eşsiz, 2019; Uzel, Saraç Eşsiz, & Özel, 2018) This problem is due to the fact that there are too many and unnecessary features in the dataset. Therefore, FSB weighting calculates very close values for features that are not in a document. To solve this problem the data is discretized by using the Weka tool (Machine Learning Group at the University of Waikato, n.d.). Data discretization is to convert continuous values into nominal values by dividing the values into specific intervals (Dougherty, Kohavi, & Sahami, 1995). Precision of feature values is improved with discretization. Discretized data is used for all classification and feature selection methods for FSB weighting.

8 different values are used for feature selection. The classification is made by selecting 3000, 2000, 1000, 500, 250, 100, 50 and 10 features, respectively. Tenfold cross-validation is used

for all classifiers. This will prevent the data from being memorized. All tables with feature selection, (Firefly+RF) and (Firefly+KNN) represent FFS that is trained with RF and KNN classifiers, respectively.

Table 4.2. Classification results for FSB and TF-IDF weightings with 10 features

Weighting+FS Methods	NB	KNN	RF	RT	RBFN
FSB+IG	87.16	99.32	100.00	100.00	94.14
TFIDF+IG	93.47	100.00	100.00	97.75	94.59
FSB +CHI2	88.51	98.20	100.00	99.32	94.14
TFIDF+CHI2	93.02	100.00	100.00	99.10	94.37
FSB+Relief-f	97.52	98.19	100.00	100.00	99.55
TFIDF+Relief-f	96.40	100.00	100.00	100.00	96.62
FSB+(Firefly+RF)	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
TFIDF +(Firefly+RF)	84.42	100.00	100.00	97.44	86.05
FSB+(Firefly+KNN)	99.32	100.00	100.00	99.77	100.00
TFIDF +(Firefly+KNN)	95.98	100.00	100.00	98.49	95.73

Table 4.2 classification results with 10 features. When we express our data with 10 features, all instances are almost identical. As a consequence, classification performances seems improved but these results are misleading.

Table 4.3. Classification results for FSB and TF-IDF weightings with 50 features

Weighting+FS Methods	NB	KNN	RF	RT	RBFN
FSB+IG	87.84	94.37	99.10	**98.20**	88.96
TFIDF+IG	85.36	**99.77**	**100.00**	97.30	88.96
FSB+CHI2	87.84	93.92	98.87	95.49	87.39
TFIDF+CHI2	85.59	**99.77**	99.32	94.37	88.51
FSB+Relief-f	87.39	94.59	98.65	96.85	**89.19**
TFIDF+Relief-f	84.01	99.55	**100.00**	95.27	84.91
FSB+(Firefly+RF)	84.23	96.62	98.87	95.49	**89.19**
TFIDF+(Firefly+RF)	91.22	**99.77**	99.77	96.62	84.91
FSB+(Firefly+KNN)	90.54	97.97	99.55	97.75	84.23
TFIDF+(Firefly+KNN)	**92.74**	98.87	**100.00**	97.05	80.50

When 50 features are selected, the data is no longer very similar. In other words, it produces an accurate result. In Table 4.3, 100% accuracy rate is achieved using the TF-IDF weighting, Firefly Feature Selection (FFS) method and Random Forest classifier. The best results, except RT, have always been achieved by FFS.

Table 4.4. Classification results for FSB and TF-IDF weightings with 100 features

Weighting+FS Methods	NB	KNN	RF	RT	RBFN
FSB+IG	87.16	91.67	97.30	94.14	86.71
TFIDF+IG	86.94	99.32	98.65	93.02	**87.84**
FSB+CHI2	86.71	92.12	95.95	**96.17**	86.94
TFIDF+CHI2	87.39	99.32	98.65	90.31	87.39
FSB+Relief-f	86.71	94.37	97.30	94.37	85.59
TFIDF+Relief-f	81.98	98.87	98.65	93.69	83.33
FSB+(Firefly+RF)	82.66	93.02	97.30	92.57	85.13
TFIDF+(Firefly+RF)	91.22	**99.55**	**98.87**	94.14	84.23
FSB+(Firefly+KNN)	83.11	88.96	95.72	93.47	78.83
TFIDF+(Firefly+KNN)	**91.44**	99.32	**98.87**	92.57	87.61

In Table 4.4, 99.55% accuracy rate is achieved using the TF-IDF weighting, FFS method, and KNN classifier. The best results, except RT and RBFN, are obtained by FFS.

Table 4.5. Classification results for FSB and TF-IDF weightings with 250 features

Weighting+FS Methods	NB	KNN	RF	RT	RBFN
FSB+IG	86.71	91.67	95.27	91.22	84.01
TFIDF+IG	87.39	97.97	95.27	88.06	**88.29**
FSB+CHI2	86.94	92.79	94.59	**93.92**	84.46
TFIDF+CHI2	87.39	97.97	96.17	91.67	88.06
FSB+Relief-f	86.26	92.34	95.49	90.99	84.01
TFIDF+Relief-f	84.23	95.95	**97.30**	91.67	85.59
FSB+(Firefly+RF)	85.81	90.31	95.04	90.09	86.94
TFIDF+(Firefly+RF)	**87.84**	98.87	96.85	89.41	86.26
FSB+(Firefly+KNN)	84.23	92.57	96.37	90.31	84.91
TFIDF+(Firefly+KNN)	82.66	**99.10**	96.62	89.86	87.61

In Table 4.5, 99.10% accuracy rate is achieved using the TF-IDF weighting, FFS method, and KNN classifier. KNN and NB classifiers have achieved good results for FFS.

According to classification results with fewer features, such as 50, 100, 250 and 500, the TF-IDF is superior to FSB. FSB needs more words to distinguish because FSB's logic is blurred. The data needs to be large for blurring. The more words that do not appear in a document, the better they will be able to express the document. In other words, if the number of documents in which the two words are common is reduced, the calculated JIs will be reduced. As a result, the weight of the word for that document will be reduced. When all JIs are zero, the weight of the word will be directly zero. As the number of features decreases, FSB weighting will change to binary weighting. 1 will be assigned as a weighting value for the words in the

document. For words that are not in the document, zero will be assigned as a weighting value. This has a negative effect on FSB's result. Since TF-IDF does not have such a constraint, it gives better results on smaller features than FSB.

Table 4.6. Classification results for FSB and TF-IDF weightings with 500 features

Weighting+FS Methods	NB	KNN	RF	RT	RBFN
FSB+IG	86.71	91.67	93.69	89.19	83.11
TFIDF+IG	**87.38**	95.49	93.02	91.89	87.61
FSB+CHI2	86.71	92.79	**95.27**	91.67	82.66
TFIDF+CHI2	86.94	95.49	94.14	89.41	86.94
FSB+Relief-f	83.78	90.54	94.14	92.12	82.21
TFIDF+Relief-f	85.59	94.59	94.37	88.51	86.26
FSB+(Firefly+RF)	84.01	90.99	94.82	90.31	83.56
TFIDF+(Firefly+RF)	84.91	**97.07**	**95.27**	**93.69**	**87.84**
FSB+(Firefly+KNN)	85.13	90.09	94.37	90.31	84.91
TFIDF+(Firefly+KNN)	86.26	96.85	94.82	87.84	**87.84**

In Table 4.6, 97.07% accuracy rate is achieved using the TF-IDF weighting, FFS method, and KNN classifier and the best results, except NB, are obtained by the FFS. Other feature selection methods are filter-based and the FFS method is a wrapper-based so the FFS method gives better results in most classifiers than others. When a firefly is running, it randomly selects features from the roulette wheel and after a certain number of iterations, a few fireflies die, replacing them with new ones. These have a good effect on the result of the firefly.

Table 4.7. Classification results for FSB and TF-IDF weightings with 1000 features

Weighting+FS Methods	NB	KNN	RF	RT	RBFN
FSB+IG	86.49	90.31	93.24	88.74	83.11
TFIDF+IG	87.39	94.14	92.79	87.16	86.71
FSB+CHI2	86.04	90.31	92.57	89.41	83.33
TFIDF+CHI2	**87.61**	**94.37**	93.02	87.39	**87.16**
FSB+Relief-f	84.23	89.19	92.57	89.19	83.11
TFIDF+Relief-f	84.91	92.79	92.57	89.19	82.66
FSB+(Firefly+RF)	84.23	90.31	94.82	**90.09**	83.11
TFIDF+(Firefly+RF)	86.04	94.14	93.02	88.29	**87.16**
FSB+(Firefly+KNN)	84.46	92.79	**96.17**	88.29	83.33
TFIDF+(Firefly+KNN)	85.13	93.92	93.02	87.84	86.71

In Table 4.7, 96.17% accuracy rate is achieved using the FSB weighting, FFS method, and RF classifier. The best results, except NB and KNN, are obtained by the FFS.

Table 4.8. Classification results for FSB and TF-IDF weightings with 2000 features

Weighting+FS Methods	NB	KNN	RF	RT	RBFN
FSB+IG	86.26	89.41	93.02	**90.54**	82.66
TFIDF+IG	88.06	**93.02**	91.44	87.16	85.59
FSB+CHI2	86.26	88.51	92.57	86.94	81.98
TFIDF+CHI2	**88.29**	92.79	91.67	86.94	85.13
FSB+Relief-f	83.56	90.31	92.34	86.49	81.98
TFIDF+Relief-f	86.04	90.77	91.89	85.59	85.36
FSB+(Firefly+RF)	84.23	91.22	**94.14**	90.31	82.21
TFIDF+(Firefly+RF)	86.49	92.79	91.67	87.16	86.04
FSB+(Firefly+KNN)	84.01	90.54	93.69	87.39	82.43
TFIDF+(Firefly+KNN)	86.26	**93.02**	92.57	87.84	**88.74**

In Table 4.8, 94.14% accuracy rate is achieved using the FSB weighting, FFS method, and RF classifier. The best results, except NB and RT, are obtained by the FFS.

Table 4.9. Classification results for FSB and TF-IDF weightings with 3000 features

Weighting+FS Methods	NB	KNN	RF	RT	RBFN
FSB+IG	86.26	89.87	92.34	**91.22**	82.21
TFIDF+IG	88.29	**92.34**	92.12	85.81	84.68
FSB+CHI2	86.71	87.84	93.02	88.74	83.78
TFIDF+CHI2	**88.51**	92.12	91.89	85.36	85.13
FSB+Relief-f	83.78	90.77	93.02	87.61	80.18
TFIDF+Relief-f	86.04	90.99	92.12	86.04	**85.81**
FSB+(Firefly+RF)	83.78	88.96	92.79	88.06	81.31
TFIDF+(Firefly+RF)	84.46	91.67	91.44	86.04	84.68
FSB+(Firefly+KNN)	84.23	88.96	**94.37**	88.29	81.98
TFIDF+(Firefly+KNN)	85.59	91.89	91.22	90.31	81.31

In Table 4.9, 94.37% accuracy rate is achieved with using the FSB weighting, FFS method, and RF classifier. RF classifier has achieved an acceptable result for FFS.

When 1000, 2000 and 3000 features are selected with the FFS method, the RF classifier always gives the best results with FSB weighting. From this point of view, training Firefly with the RF classifier raises the question of whether the results are misleading in favor of the RF and FSB. In response to this question, it is decided to train FFS with the KNN classifier to see the difference in results. In this way, it can be seen that FSB gives the best results again. It is understood that without the influence of the RF classifier, the FSB continues to give good results. As the number of features increases, FSB gives better results than TF-IDF. This confirms the thesis that was defended in the first place.

Table 4.10. Runtime for classifiers according to the different number of features (seconds)

Number of features	NB	KNN	RF	RT	RBFN
10	0.0000001	0.0000001	0.02	0.0000001	0.01
50	0.0000001	0.0000001	0.05	0.0000001	0.01
100	0.0000001	0.0000001	0.06	0.0000001	0.02
250	0.01	0.0000001	0.12	0.0000001	0.05
500	0.02	0.0000001	0.15	0.0000001	0.09
1000	0.02	0.0000001	0.25	0.0000001	0.15
2000	0.06	0.0000001	0.43	0.01	0.33
3000	0.08	0.0000001	0.56	0.01	0.42
8308	0.24	0.0000001	1.72	0.02	1.16

Table 4.10 shows the run time of the classifiers in seconds. The values shown in the table are taken as a result of the TF-IDF weighting and IG feature selection method. Other classifiers and weighting methods have not been calculated since they will take approximately the same timing. Classification runtimes are decreased and classification accuracy is increased as the number of features decreases regardless of weighting and feature selection methods.

5. CONCLUSIONS

5.1. Summary

Because Android is an easy and open source operating system, the usage of Android phones is increased day by day. As a result, it is also the target of malicious people. There are applications installed on Android by Google Play Store or 3rd party application providers. The security of these applications is provided by the Google Play Store in certain ways, but this is not enough. For this, various studies have detected malicious software in android applications as static and dynamic methods. In this study, the static analysis method is used to determine whether the APK is malware or benign. For this, TF-IDF and FSB weighting methods are used. The FSB weighting method has not been used in other studies to detect malware. NB, KNN, RF, RT, and RBFN which are the most known machine learning classifiers are used. In addition to IG, CHI2, Relief-f feature selection methods, a new method, the FOA is applied as a feature selection method. Nature inspired algorithms have not been used very much when choosing features to detect malware. This study aims to investigate the contribution of FSB and FFS as a new and different method for malware detection. Without feature selection, the FSB weighting method yields better results than the TF-IDF weighting method, whereas, after the feature selection, both methods are mutually superior. As the number of features decreases, the data is completely separated from each other. According to FSB weighting logic, the FSB weighting method looks like binary weighting. For these reasons, as the number of features decreases, TF-IDF will give better results than FSB. Also, the best results are always obtained with the FFS method.

5.2. Contributions and Results

It is clear that FSB weighting gives good results compared to TF-IDF before selecting features. After selecting features, they provide superiority to each other in a different number of features. In our previous studies with FSB, we knew that FSB gave good results according to TF-IDF. However, in this study, as the number of features decreases, it is determined that FSB will turn into binary weighting. Therefore, it is concluded that the decrease in the number of features will not be a good choice for FSB. In addition, the FFS method has achieved better results according to the other feature selection methods and it has the best results in most classifiers. Because FFS is a wrapper method, it gives better results. Moreover,

various randomizations implemented in FFS take it to a better place. Since it is influenced by the classifier it is trained in, it is negatively affected at some points. Another point of influence is that even if randomization often improves, sometimes it may not. For this reason, it has achieved very good results at most points, and the results have been negatively affected at a few points.

6. RECOMMENDATIONS

Only static features are used. Therefore, the data instances will be similar to each other as the number of features decreases. This constraint can be overcome by adding dynamic features in future studies. In FSB, the computational load is a little heavy. This affects the runtime. A few improvements can be offered to shorten runtime on FSB in future studies. In future studies, a classification can be made with malware types and important features can be determined for the type of malware. Also, in the FFS method, weights of features can be given by a different method instead of the Relief-f method. The f-measure value used for FFS can be tried with more classifiers and results can be obtained. Besides, algorithms can be used to optimize the parameters used in the FFS method.

REFERENCES

Adebayo, O., & AbdulAziz, N. (2014). Android Malware Classification Using Static Code Analysis and Apriori Algorithm Improved with Particle Swarm Optimization. *4th World Congress on Information and Communication Technologies (WICT 2014).*

Aha, D.W., Kibler, D. and Albert, M.K. (1991) Instance-based learning algorithms. *Machine Learning, 6*, 37-66.

Alther, A., & Barukab, O. (2017, May 29). Android malware classification based on ANFIS with fuzzy c-means clustering using significant application permissions. *Turkish Journal of Electrical Engineering & Computer Sciences*.

Al-Sheshtawi, K. A., Abdul-Kader, H. M., & Ismail, N. A. (2010). Artificial Immune Clonal Selection Classification Algorithms for Classifying Malware and Benign Processes Using API Call Sequences. *International Journal of Computer Science and Network Security (IJCSNS), 10*(4).

Anbu, M., & Anandha Mala, G. S. (2019). Feature selection using firefly algorithm in software defect prediction. *Cluster Computing, 22*. doi:10.1007/s10586-017-1235-3

Anderson, B., Quist, D., Neil, J., Storlie, C., & Lane, T. (2011). Graph-based malware detection using dynamic analysis. *7*, 247-258.

Android Developers. (n.d.). Retrieved from https://developer.android.com/guide/platform

Android Developers. (n.d.). *Package Index*. Retrieved from https://developer.android.com/reference/packages.html

Anupriya, A., Nithya, V., & Ponmalar, S. (2018, March). A Survey: Analysis of Virus and Malware Detection. *International Journal of Innovative Research in Computer and Communication Engineering, 6*(3).

Bazrafshan, Z., Hashemi, H., Fard, S., & Hamzeh, A. (2013). A Survey on Heuristic Malware Detection Techniques. *5th Conference on Information and Knowledge Technology (IKT).*

Breiman, L. (2001). Random Forests. *Machine Learning 45*(1), 5–32.

Broomhead, D. S., & Lowe, D. (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks (Technical report). *RSRE*. 4148.

Broomhead, D. S., & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems, 2*, 321–355.

Castillo, C. (2011). Android Malware Past, Present, and Future.

Chen, J., Alalfi, M. H., Dean, T. R., & Zou, Y. (2015). Detecting Android Malware Using Clone Detection. *Journal of Computer Science and Technology, 30*, 942-956.

Çarkacı, N., & Soğukpınar, İ. (2016). Frequency-based metamorphic malware detection. *24th Signal Processing and Communication Application Conference (SIU)*, (pp. 421-424). Zonguldak. doi:10.1109/SIU.2016.7495767

De La Rosa, L., Kilgallon, S., Vanderbruggen, T., & Cavazos, J. (2018). Efficient Characterization and Classification of Malware Using Deep Learning., (pp. 77-83). doi:10.1109/RWEEK.2018.8473556

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and Unsupervised Discretization of Continuous Features.

Drake, N., & Turner, B. (2019, July 24). *Best Android antivirus apps*. Retrieved from https://www.techradar.com/best/best-android-antivirus-app

Firdausi, I., Lim, C., Erwin, A., & Nugroho, A. S. (2010, December 2-3). ANALYSIS OF MACHINE LEARNING TECHNIQUES USED IN BEHAVIOR-BASED MALWARE

DETECTION. *Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*.

G., N. (2019, January 28). *Android Statistics - Growth And Market Share*. Retrieved from https://techjury.net/stats-about/android-market-share/

Gadhiya, S., & Bhavsar, K. (2013, April). Techniques for Malware Analysis. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(4).

Gutzman, C., Sweep, S., & Tambo, A. (2003). Differences and Similarities of Spyware and Adware. *Computer Science Seminar Fall*.

Han, J., & Kamber, M. (2006). *Data Mining: Concepts and Techniques*. San Francisco: Morgan Kaufmann Publishers.

Ho, T. K. (1995). Random Decision Forests. *Proceedings of the 3rd International Conference on Document Analysis and Recognition.* Montreal.

Jeyarani, D. S., & Pethalakshmi, A. (2016, August). Optimized Feature Selection Algorithm for High Dimensional Data. *Indian Journal of Science and Technology, 9*(26). doi:10.17485/ijst/2016/v9i31/79656

Kira, K. & Rendell, L. (1992). The Feature Selection Problem: Traditional Methods and a New Algorithm. *AAAI-92 Proceedings*.

Kira, K. & Rendell, L. (1992). A Practical Approach to Feature Selection. *Proceedings of the Ninth International Workshop on Machine Learning*, p249-256.
Kononenko, I. (1994). Estimating attributes: Analysis and extensions of RELIEF. In *Machine Learning: ECML-94* (pp. 171-182). Springer Berlin Heidelberg.

Lashkari, A. H., A. Kadir, A. F., Laya, T., & Ghorbani, A. A. (2018). Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification.

52nd IEEE International Carnahan Conference on Security Technology (ICCST). Montreal, Quebec, Canada.

Machine Learning Group at University of Waikito. (n.d.). *Weka 3 - Data Mining with Open Source Machine Learning Software*. Retrieved from https://www.cs.waikato.ac.nz/ml/weka/

Maron, M. E. (1961). Automatic Indexing: An Experimental Inquiry. *Journal of the ACM, 8(3)*, 404–417. doi:10.1145/321075.321084

Mathur, K., & Hiranwal, S. (2013, April). A Survey on Techniques in Detection and Analyzing Malware Executables. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(4).

Milošević, N. (2013). History of malware. *CoRR*, 1-11. Retrieved from http://arxiv.org/abs/1302.5392

Ni, Z., Yang, M., Ling, Z., Wu, J., & Luo, J. (2016). Real-Time Detection of Malicious Behavior in Android Apps. *International Conference on Advanced Cloud and Big Data (CBD).* Chengdu.

Obeis, N. T., & Bhaya, W. (2018). Malware Analysis Using Apis Pattern Mining. *International Journal of Engineering & Technology, 7*, 502-506.

Parkour, M. (n.d.). *contagio mobile*. Retrieved from http://contagiominidump.blogspot.com/

Pektaş, A., Pektaş, E. N., & Acarman, T. (2018). MINING PATTERNS OF SEQUENTIAL MALICIOUS APIS TO DETECT MALWARE. *International Journal of Network Security & Its Applications (IJNSA), 10*(4).

Puram, N. M., & Singh, K. R. (2018, April). An Implementation to Detect Fraud App Using Fuzzy Logic. *International Journal on Future Revolution in Computer Science & Communication Engineering, 4*(4).

Razak, M. F., Anuar, N. B., Othman, F., Firdaus, A., Afifi, F., & Salleh, R. (2018). Bio-inspired for Features Optimization and Malware Detection. *Arabian Journal for Science and Engineering*. doi:10.1007/s13369-017-2951-y

Rodríguez-Mota,, A., Escamilla-Ambrosio, P., & Salinas-Rosales, M. (2017). Malware Analysis and Detection on Android: The Big Challenge. doi:10.5772/intechopen.69695

Saha, B., & Gairola, A. (2005, June 17). Botnet: An Overview. *CERT-In White Paper CIWP-2005-05*.

Salton, G., & Buckley C (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management, 24*(5), 513-523.

Santos, I., Penya, Y., Devesa, J., & Bringas, P. (2009). N-GRAMS-BASED FILE SIGNATURES FOR MALWARE DETECTION., (s. 317-320).

Saxe, J., & Berlin, K. (2015). Deep neural network based malware detection using two dimensional binary program features. *10th International Conference on Malicious and Unwanted Software (MALWARE)*, (pp. 11-20). Fajardo. doi:10.1109/MALWARE.2015.7413680

Sharma, A., & Sahay, S. (2015, October). Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey.

Soner, S., Soner, S., & Yaday, M. (2015). A Survey on Software Bug Evaluation. *International Journal of Computer Applications, 129*(10).

Tailor, J., & Patel, A. (2017). Comprehensive Survey: Ransomware Attacks Prevention, Monitoring and Damage Control. *International Journal of Research and Scientific Innovation (IJRSI), IV*(VIS), 2321-2705.

Turner, A. (2019, October). *How Many People Have Phones Worldwide*. Retrieved from https://www.bankmycell.com/blog/how-many-phones-are-in-the-world

Uppal, D., Mehra, V., & Verma, V. (2014, February). Basic survey on Malware Analysis, Tools and Techniques. *International Journal on Computational Sciences & Applications (IJCSA), 4*(1).

Uzel, V. N., & Saraç Eşsiz, E. (2019). Detecting Sms Spam with Fuzzy Weighting Method. *International Conference on All Aspects of Cyber Security 2019 (A2CS'19).* Adana.

Uzel, V. N., Saraç Eşsiz, E., & Özel, S. A. (2018). Using Fuzzy Sets for Detecting Cyber Terrorism and Extremism in the Text. *2018 Innovations in Intelligent Systems and Applications Conference (ASYU).* doi:10.1109/ASYU.2018.8554017

Wang, Y., Wen, S., Xiang, Y., & Zhou, W. (2014). Modeling the Propagation of Worms in Networks: A Survey. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS, 16(2)*.

Wei, F., Li, Y., Roy, S., Ou, X., & Zhou, W. (2017). Deep Ground Truth Analysis of Current Android Malware. *14th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2017).* Bonn.

Yang X. (2010). *Nature-Inspired Metaheuristic Algorithms*. United Kingdom: Luniver Press.

Yates, F. (1934). Contingency table involving small numbers and the $\chi 2$ test. *Supplement to the Journal of the Royal Statistical Society, 1*(2), 217–235.

Yim, K., Kim, S., Park, J., Lee, K., & You, I. (2012). A Brief Survey on Rootkit Techniques in Malicious Codes. *Journal of Internet Services and Information Security (JISIS), 3*(4), 134-147.

Yuan, Z., Lu, Y., Wang, Z., & Xue, Y. (2014). Droid-Sec: Deep Learning in Android Malware Detection. *ACM SIGCOMM Computer Communication Review*. doi:10.1145/2619239.2631434

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, 8*(3), 338-353.

Zhenfang, Z. (2015, August). Study on Computer Trojan Horse Virus and Its Prevention. *International Journal of Engineering and Applied Sciences (IJEAS), 2*(8).

CURRICULUM VITAE

Vahide Nida UZEL was born in Adana, TURKEY, in 1994. She studied in Bialystok, Poland with Erasmus+ program in 2016. She graduated from the Computer Engineering Department of Çukurova University with a second degree in 2017. She started her master's degree in Computer Engineering at Çukurova University in 2017. She started to work as a research assistant in the Computer Engineering Department at Adana Alparslan Türkeş Science and Technology University in March 2018. She gave up her master's degree in Çukurova University and started her master's degree in Cyber Security Department at Adana Alparslan Türkeş Science and Technology University in September 2018. Her current research interests include malware detection, nature inspired algorithms, machine learning, and data mining.