الجمهورية الجزائرية الديمقراطية الشعبية

People's Democratic Republic of Algeria

وزارة التعليم العالي والبحث العلمي

Ministry of Higher Education and Scientific Research

جامعة غرداية

University of Ghardaïa

كلية العلوم والتكنولوجيا

Faculty of Science and Technology

قسم الرياضيات والإعلام الآلي

Department of Mathematics and Computer


***Registration No.***
/ … / … / … / … / …

# Thesis

**To obtain the master's degree**

**Field:** Mathematics and Computer Science
**Faculty:** Science and Technology
**Specialty:** Intelligent Systems for Knowledge Extraction

**Theme**

# Machine Learning for smartphone security: Android botnet detection


By
Bayoub REDDAH
Nessreddin DAOUDI


**Before the jury composed of:**

| | | | |
|---|---|---|---|
| Slimane OULAD-NAOUI | MCB | Univ. Ghardaia | Examiner |
| Houssem Eddine DEGHA | MAA | Univ. Ghardaia | Examiner |
| Nacera BRAHIM | MAA | Univ. Ghardaia | Supervisor |

University Year 2020 / 2021

## Abstract

Android is the most used mobile operating system in the world and since it is open source, hackers exploit it to perform different attacks such as executing botnet attack which allow them to control the compromised device remotely from a Command and control (C&C) server and perform other attacks such as distributed denial of service (DDOS) from the device itself without the owners' knowledge.

The aim of our study is to find a model that allows us to detect Android botnets efficiently.

Our proposed method uses a single layer and multi-layer Perceptron models trained on 342 features to classify application as benign or botnet using ICSX dataset.

We obtained great results from our experimental study with an accuracy of 99%.

**Keywords:** Botnet detection, Android Botnets, Mobile Botnet, Machine learning, Perceptron, Multi-layer Perceptron, Static Analysis, Smartphone Security .

**Résumé**

Android est le système d'exploitation mobile le plus utilisé au monde du fait qu'il est open source, raison pour laquelle les pirates tentent de plus en plus de l'exploiter pour lancer différentes attaques telles que la mise en place d'un botnet qui leur permet de contrôler l'appareil compromis à distance à partir d'un serveur Command and control (C&C) et ainsi s'en servir comme support pour lancer d'autres attaques telles que le déni de service distribué (DDOS) à partir de l'appareil lui-même à l'insu de son propriétaire.

L'objectif de notre étude est de trouver un modèle qui nous permet de détecter les botnets Android de manière efficace.

Notre solution a été implémentée en utilisant des Perceptron à couche unique et multicouches qui a été entrainé sur 342 caractéristiques pour distinguer les applications bénignes des botnet en se servant de la base de données (corpus d'entrainement) ICSX.

Les résultats obtenus de notre éxperimentaion sont très encourageants avec un taux de précision de teste qui a atteint 99%.

**Mots clés :** Détection des botnets, botnets Android, botnet mobile, apprentissage automatique, Perceptron, Perceptron multicouches, analyse statique, sécurité des smartphones

**ملخص**

أندرويد هو نظام التشغيل الأكثر استخداما في العالم، ونظرا لكونه مفتوح المصدر، يحاول المخترقين إستغلاله لشن مختلف الهجمات مثل هجوم Botnet والذي يمكنهم من التحكم في جهاز الضحية عن بعد باستخدام سيرفير التحكم (Command & Control) حيث يمكنهم من شن هجمات أخرى مثل هجوم (Distributed Denial of Service) من الجهاز نفسه بدون علم مالك الجهاز.

هدفنا هو إيجاد طريقة تسمح لنا باكتشاف تطبيقات Botnet على نظام الأندرويد بكفاءة عالية.

طريقتنا المقترحة تستتخدم نموذج Perceptron ذو طبقة واحدة و ذو عدة طبقات الذي تم تدريبه على 342 خاصية لتصنيف التطبيقات على انها سليمة أو Botnet باستخدام مجموعة البيانات ICSX.

تحصلنا على نتائج رائعة من خلال دراستنا التجريبية بدقة 99%

**الكلمات المفتاحية**: الكشف عن الشبكات الروبوتية، الشبكات الروبوتية لنظام الأندرويد، الشبكات الروبوتية للهاتف المحمول، التعلم الألي، بيرسبترون، متعدد الطبقات بيرسبترون، تحليل ثابت، أمان الهواتف

# Dedication

We thank Allah who helped us achieve this accomplishment and he has been with us from the beginning.

We dedicate this work to our family and our friends. A special feeling of gratitude to our loving parents, brothers and sisters, who have been a constant source of support and encouragement.

## Acknowledgement

First, we thank Allah for helping us complete this thesis.

Next, we would like to express our special gratitude to our supervisor Mrs. Nacera BRAHIM, for her follow-up, guidance and support, Which helped us in writing this thesis with her suggestions and encouragement.

Moreover, we would also like to thank with great appreciation Dr. Selimane BELLAOUAR as well as Dr. Selimane OULADNAOUI for their assistance and guidance in this paper.

Finally, we would like to express our deep appreciation to everyone that helped us complete this report without forgetting our teachers who taught us since the first grade because we wouldn't be here without them.

# Contents

# List of Figures

# List of Tables

# List of abbreviations and acronyms

| | |
|---|---|
| **API** | *Application Programming Interface* |
| **SVM** | *Support Vector Machine* |
| **ROC** | *Receiver operating characteristic* |
| **CSV** | *Comma-separated values* |
| **PSO** | *Particle Swarm Optimization* |
| **BRF** | *Radial Basis Function* |
| **OS** | *Operating System* |
| **C&C** | *Command and Control* |
| **DDoS** | *Distributed Denial of Service* |
| **CNNs** | *Conventional Neural Networks* |
| **ART** | *Android RunTime* |
| **HAL** | *Hardware Abstraction Layer* |
| **APK** | *Android package kit* |
| **AI** | *Artificial Intelligence* |
| **ML** | *Machine Learning* |

| | |
|---|---|
| **DL** | *Deep Learning* |
| **P2P** | *Peer to Peer* |
| **IRC** | *Internet Relay Chat* |
| **HTTP** | *Hypertext Transfer Protocol* |
| **CPU** | *Central Processing Unit* |
| **RAM** | *Random-Access Memory* |
| **SMS** | *Short Message Service* |
| **ADB** | *Android Debug Bridge* |
| **TCP** | *Transmission Control Protocol* |
| **ICMP** | *Internet Control Message Protocol* |
| **UDP** | *User Datagram Protocol* |
| **NB** | *Naïve Bayes* |
| **KNN** | *K-Nearest Neighbors* |
| **RF** | *Random Forest* |
| **ISCX** | *Information Security Center of Excellence* |
| **IDC** | *International Data Corporation* |
| **DNS** | *Domain Name System* |
| **AAPT** | *Android Asset Packaging Tool* |
| **SMO** | *Sequential Minimal Optimization* |
| **SLR** | *Simple Logistic Regression* |

| | |
|---|---|
| **IG** | *Information Gain* |
| **CV** | *Cross Validation* |
| **ANNs** | *Artificial Neural Networks* |
| **MLP** | *Multi-Layer Perceptron* |
| **ReLU** | *Rectified Linear Unit* |
| **Tanh** | *Hyperbolic tangent* |
| **GAN** | *Generative Adversarial Network* |
| **NLP** | *Natural Language Processing* |
| **PCA** | *Principal Component Analysis* |
| **MitM** | *Man in the Middle* |

# Introduction

Smartphones are advanced mobile devices that help users perform their daily tasks faster, however doing so means that most of their private information is stored on their smartphones, which are running usually on iOS or Android operating system.

Android is the most dominant operating system in the smartphone market, and in 2020 the market share of android is 72.72% followed by iOS with a share of 26.47%. [1]

Android has a high market share because it is an open source operating system, but that allows hackers to find vulnerabilities easily, which helps them to develop advanced malware to attack it. According to McAfee's Threat Report for 2021 [2] the number of new malware on mobile devices has increased to 3.4 million as of the fourth quarter of 2020. The advanced malware can be used to execute botnet attack which allows them to control the compromised device remotely from a Command and control (C&C) server to perform other attacks such as distributed denial of service (DDOS) from the device itself without the owners' knowledge. So we need efficient methods that can detect Android botnet applications.

The rest of the thesis will be organized as follows: The first chapter provides basic information about the Android operating system and the major attacks on it. It also provides some basic information about machine learning techniques and their metrics. The second chapter explains botnets and their attacks. The third chapter presents the latest methods that can detect Android botnet applications. Finally, the last chapter explains our proposed method for detecting Android botnet applications.

# Chapter 1

# Background

## 1.1 Introduction

Android is among the most used operating systems because it is an open source system but that makes it vulnerable to multiple attacks and to fight that. Artificial intelligence systems are used because they can detect new attacks better than other methods.

In this chapter we present Android operating system architecture, the different existing attacks, and machine learning techniques with their metrics.

## 1.2 Android operating system

### 1.2.1 Definitions

**Android** is an open source operating system based on Linux that can be used on different devices and form factors. [3]

**Android Package Kit (apk)** is a file format used to distribute and install android applications, it usually contains the following files [4]:

- AndroidManifest.xml file which contains the application information such as application name, required permissions, broadcast receivers, intents.

- classes.dex file which contains the application source code compiled in the dex file (a bytecode format created specially for Android and it is optimized to reduce memory usage).
- lib directory which contains compiled code for specific platforms such as armeabi-v7a, arm64-v8a, ...etc.
- res directory which contains non-compiled resources.
- assets directory which contains applications assets.
- resources.arsc file which contains pre-compiled resources.

### 1.2.2 Platform architecture

The architecture of the Android system is shown in figure 1.1 and according to [3] the major components of an Android system are:

- **Linux Kernel** is the foundation of the Android platform that helps Android to take advantage of key security features while allowing device manufacturers to develop hardware drivers for a well-known kernel.
- **Hardware Abstraction Layer (HAL)** provides a Java API interface to the device's hardware, it contains library modules for each type of hardware such as Audio, Bluetooth, ...etc.
- **Android Runtime** it can run multiple virtual machines on the device by executing DEX files (a bytecode format created specially for Android and it is optimized to reduce memory usage).
- **Native C/C++ Libraries** is a set of libraries written in C and C++ that can be accessed directly from native code in any application.
- **Java API Framework** is an application interface to all Android OS features written in Java, it allows app developers to write simple and reusable code.
- **System Apps** is a set of core apps that allows users to have basic functionalities pre-installed into their phone while allowing app developers to use Android OS key features without the need to write code from scratch.

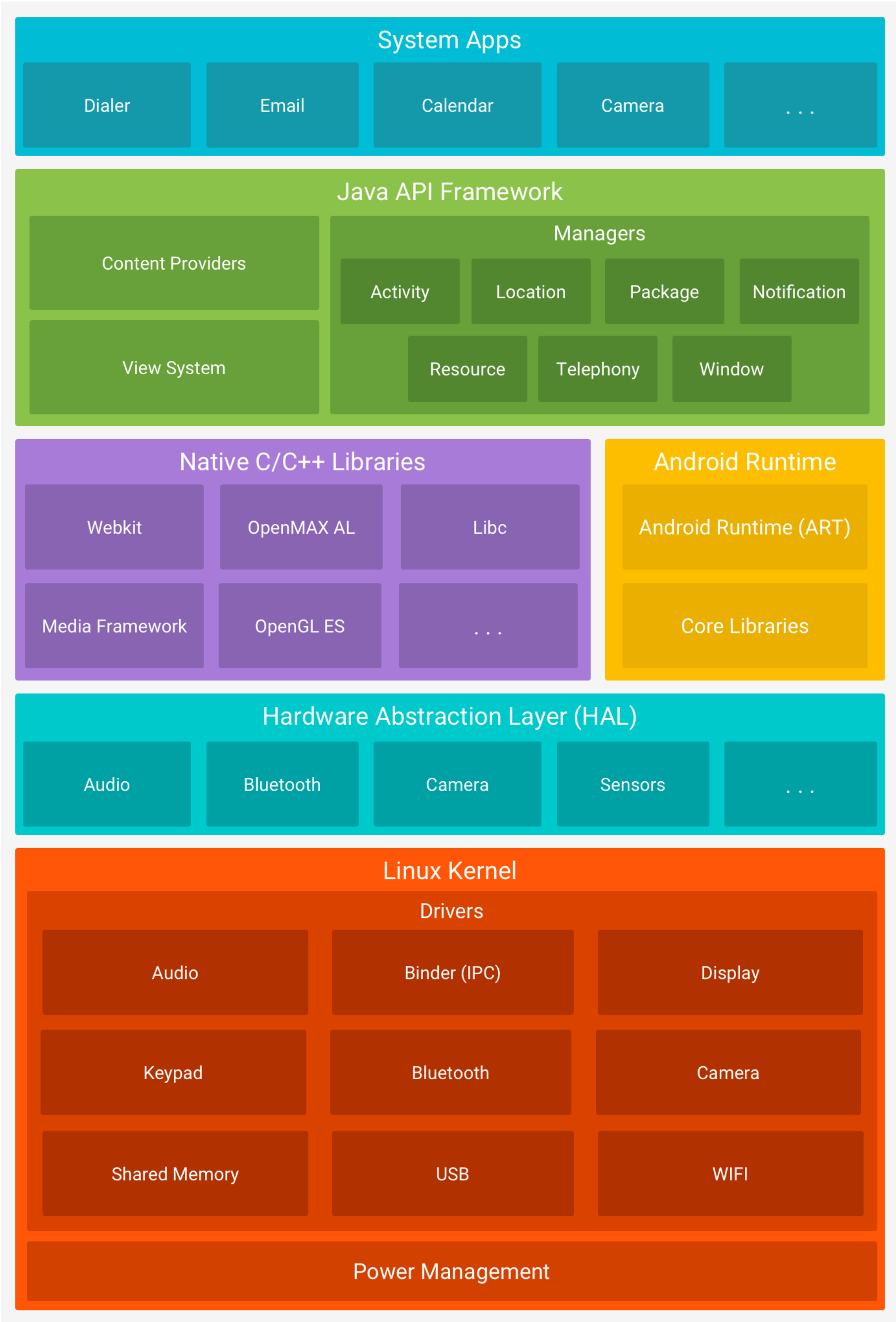



Figure 1.1: Android platform architecture [3]

### 1.2.3 Risks

John Chambers, the former CEO of Cisco, once said: "*There are two types of companies: those that have been hacked, and those that don't yet know that they have been hacked*". According to Cisco's annual Cybersecurity Report, the total number of attacks has nearly quadrupled between January 2016 and October 2017 [5].


Figure 1.2: Types of cyber attacks [6]

According to [5, 7, 8] there are several methods and many types of electronic attacks. Among the most well-known attacks are the following:

**Malware** is every software with bad intent such as spyware, ransomware, viruses, Trojans, and worms. In other words, it is malicious software programs that when they get installed into the victim's system they can send the victim's data to the hacker, lock the victim's files, serve fraud advertisement, divert traffic, sniff the victim's data, spread to other devices, etc.

**Trojans** are different than viruses and worms because they are not meant to damage or delete files on your system. Their principal task is to provide a backdoor gateway for malicious programs/users to steal your valuable data without your knowledge and permission.

**Viruses** have the ability to replicate themselves and they damage files on the victim's device. They stick themselves to songs, videos, and executable files and travel all over the internet. Their main weakness lies in the fact that viruses can get into action only if they have the support of a host program. W32.Sfc!mod, ABAP.Rivpas.A, Accept.3773 are some examples of virus programs.

**Phishing** is a hacking technique used by hackers to replicate the most accessible websites and traps the victims by sending a spoofed link to the replicated website. Along with social engineering, it becomes one of the most common and most lethal attacks. The primary objective of the attack is to steal sensitive victim's data such as credit card and login information or install malware on the victim's device.

**Denial of Service (DoS)** A denial of service attack is a technique used to take down a site, network, or server by flooding that site or server with so much traffic that the server cannot process all requests in real time and finally crashes, The attacker floods the target machine with tons of requests to flood the resources, which in turn limits the fulfillment of legitimate clients requests.

**Distributed Denial of Service (DDoS)** For DDoS attacks, attackers can use several compromised devices to launch this attack, so often hackers deploy botnets or zombie devices that have the sole act of flooding the target system with requests. The scale of DDoS attacks is increasing with each passing year.

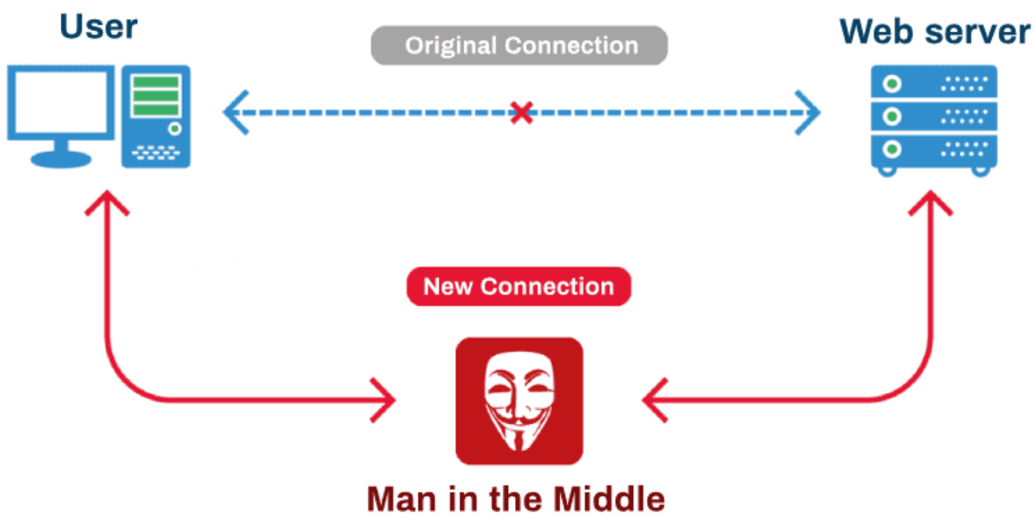



Figure 1.3: Man In The Middle attack [9]

**Man-in-the-middle (MitM)** the attacker inserts himself between the communication of two devices to read the nonencrypted traffic without alerting the involved devices as shown in figure 1.3, which allows him to steal financial details and private information.

## 1.3 Machine Learning & Deep Learning

According to [10, 11] the field of Machine Learning is used to solve many problems such as identifying spam, making product recommendations, and forecasting demand. Deep Learning (DL) is part of Machine Learning which is part of Artificial Intelligence as shown in Figure 1.4.

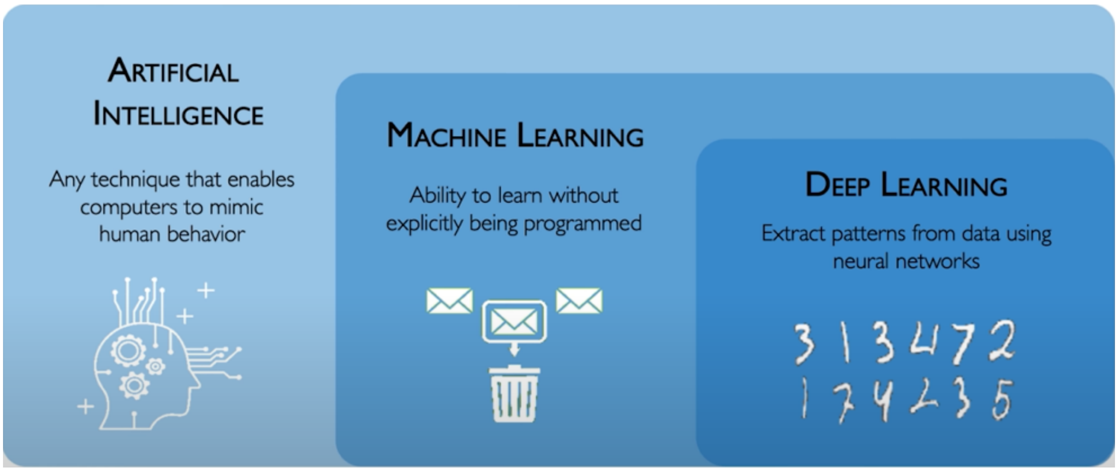



Figure 1.4: The difference between (AI-ML-DL) [12]

Machine learning is more expert and innovative compared to artificial intelligence, as machine learning is used to solve many big problems such as: making product recommendations, customer segmentation, demand forecasting, identifying spam, categorizing news articles according to their fields (politics, sports, economics...), ...etc.

Machine learning consists of several algorithms such as: Naïve Bayes Classification(NB), Decision Trees, Logistic Regression, linear Regression, Particle Swarm Optimization (PSO), Support Vector Machines (SVM), Clustering Algorithms, Principal Component Analysis (PCA), etc. Some of them are shown below:

**Support Vector Machine (SVM):** is a supervised learning model that uses kernel tricks such as Radial Basis Function (RBF) kernel, Sigmoid kernel, Polynomial kernel, etc. SVMs create a set of hyper-planes in a infinite dimensional space, which can be used for classification and regression problems. When the hyper-plane has the greatest distance to the nearest training data points of any category (functional margin) good separation is achieved. The generalization error of the classifier is lower if the margin is greater. Figure 1.5 shows the decision function for a linearly separable problem, having several samples on the margin boundaries (support vectors). [13]

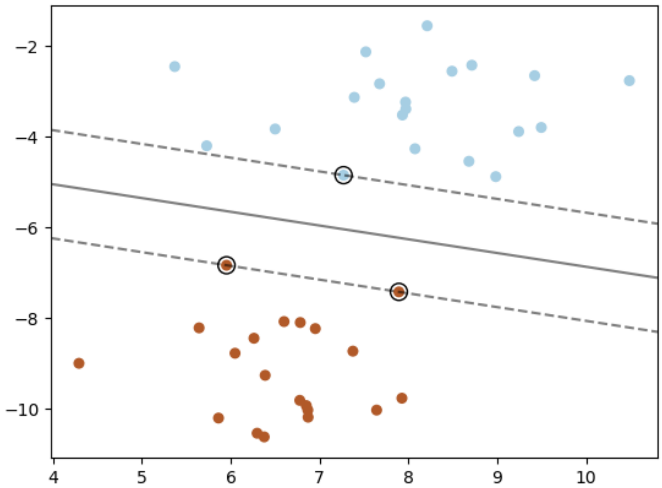



Figure 1.5: SVM hyper-planes [13]

**Random forest (RF):** is a supervised learning algorithm. It constructs a forest which is a set of decision trees that are trained independently on data's random subset, usually trained using the "bagging" method which is a random sampling technique with replacement. [14] Figure 1.6 shows an example of a decision tree.

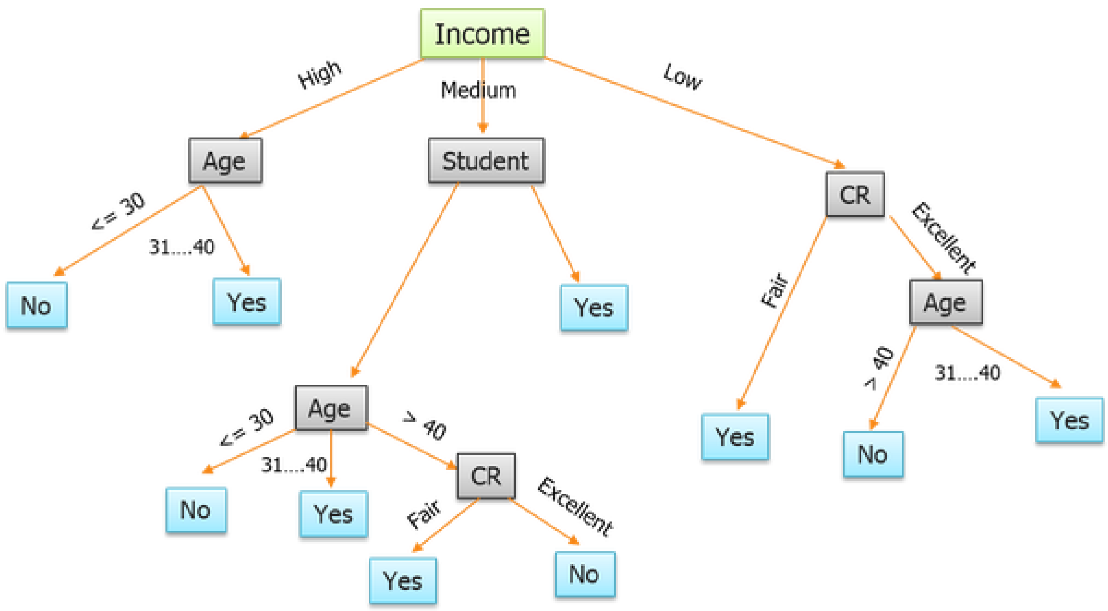



Figure 1.6: Decision tree method example [15]

**Particle Swarm Optimization (PSO):** Eberhat and Kennedy [16] introduced a method to solve optimization problems after they observed the social behavior of birds and schools of fish. The main advantages of this method is the fact that it converges quickly to the global best point, has a simple execution, has only few of adjustable parameters, it uses very little computation power and it can find the optimal solution for continuous and discrete mathematical problems. [17]

### 1.3.1 Architectures

**Perceptron**: the perceptron is the name given by the neuroscientist Frank Rosenblatt to a group of experiments that he began to simulate the human mind in the thought process between 1957-1962, and it led to his creation of the first Artificial Neural Network(ANN) in history, the Perceptron Neural Network has only the input and output layer, and uses the Heaviside step activation function on the output node which is defined as follows:

$$h(x) = \begin{cases} 1, & w * x + b > 0 \\ 0, & \text{otherwise} \end{cases}$$

Perceptron neural Network is a supervised learning algorithm, and a linear binary classifier which means that the network solves problems that can only be separated in a linear form, after the original model was introduced many updated models emerged.

Deep learning models are more accurate than traditional machine learning algorithms however, it requires a large datasets. Deep learning is used to solve many complex problems such as speech recognition, computer vision, autonomous driving, and natural language processing (NLP).

Every deep learning model uses activation functions which are responsible for calculating the sum of the product of weights with different inputs in a given range to determine the value of the final output of the current layer, which will be the inputs for the next if not the last layer. They are used to get the output of the node, in the design of the neural network, the activation functions are an important part of it and some of the popular activation functions are: Identity, Binary step, sigmoid, Hyperbolic tangent(tanh), Rectified linear unit (ReLU), Leaky rectified linear unit (Leaky ReLU), Softmax, etc. The ability and performance of the neural network is affected by the choice of the activation function, Therefore, a careful selection of the activation function must be made.

Deep learning consists of several algorithms such as Artificial Neural Network(ANN), Convolutional Neural Network(CNN), Multi-Layer perceptron (MLP), Recurrent neural network(RNN), Generative Adversarial Network(GAN), etc. some of which are explained below:

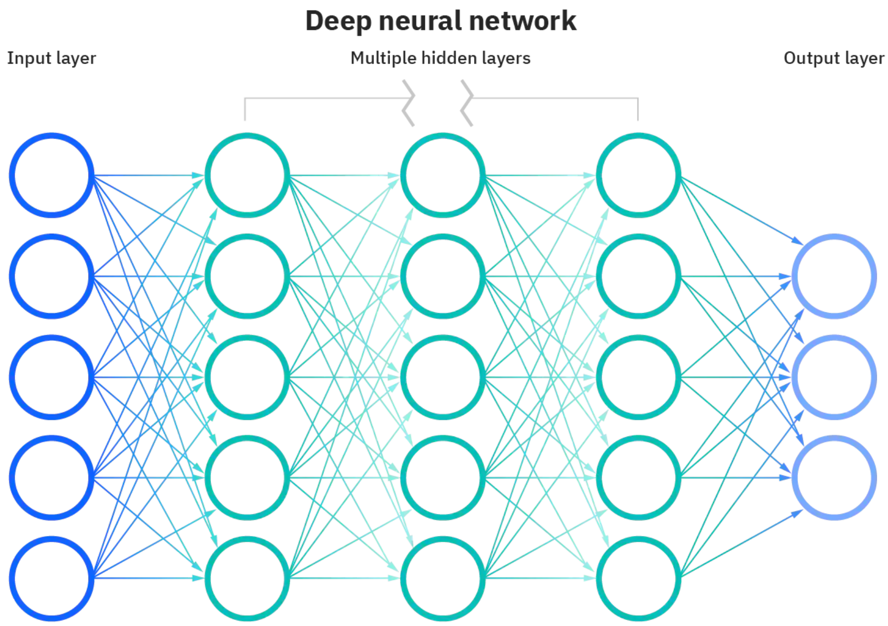



Figure 1.7: Deep neural network [18]

- **Artificial Neural Network (ANN)**: it simulates the human brain by learning from observational data. the figure 1.7 shows the architecture of a deep neural network. The difference between artificial neural networks and deep learning networks is the depth of the hidden layers in the neural network.

  **Multi-Layer Perceptron (MLP):** is a class of feedforward Artificial Neural Networks (ANN), consisting of an input layer and an output layer like a perceptron with other layers between them which are called hidden layer(s), MLP uses the Backpropagation method to update the weights of neurons, however this method requires non-linear activation functions such as ReLU, Sigmoid, ...etc, due to that MLP can solve non-linear problems and the main use cases for MLP are prediction, identification, and classification.

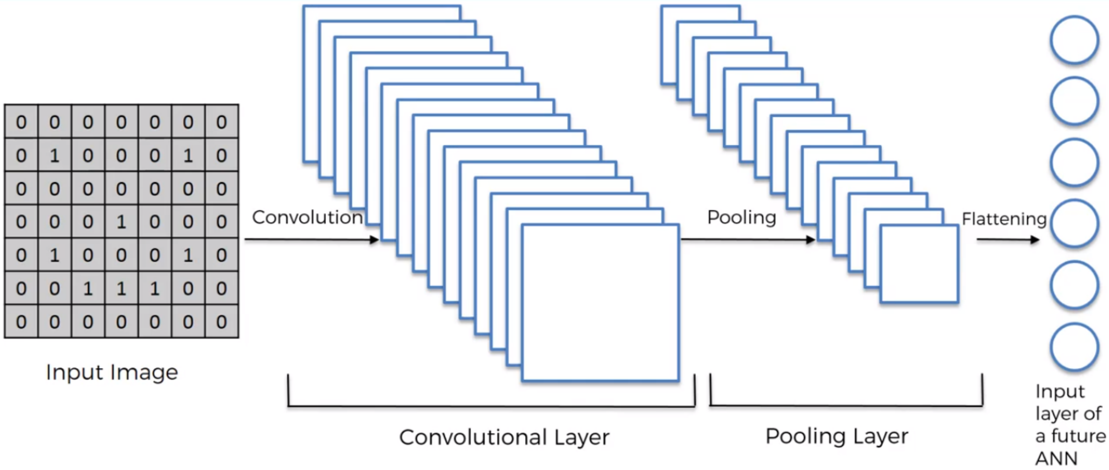



Figure 1.8: CNN architecture [19]

- **Convolutional Neural Network (CNN)**: is a neural network that are mainly used to solve image and video recognition problems by extracting the high-level features, it consists of three distinct operational layers : convolutional layer, pooling layer, fully connected layer, as shown in figure 1.8.

### 1.3.2 Evaluation

According to [20–22] we can verify and evaluate our model using some techniques such as:

**Cross-Validation (CV):** is a statistical method used to test and estimate the effectiveness of machine learning models.

The Cross Validation's goal is to test the ability of model to predict new data that was not used in estimating it, so as to solve problems such as selection bias or overfitting and give an insightful vision to how the model will generalize to an independent dataset. Cross-validation's One of the tours involves Splitting a sample of data into complementary subsets, proceeding the analysis on one subset (training set), and validation of the analysis over the other subset (testing set). To decrease variability, mostly, cross-validation's multiple rounds are made using different partitions, and the validation results are combined (e.g. median) over the rounds to give an estimate the predictive performance of the model. [23]

We can also study the performance of a classification model by calculating the model metrics using the confusion matrix which is a table layout that describes the performance of a classification model, usually a supervised learning one.

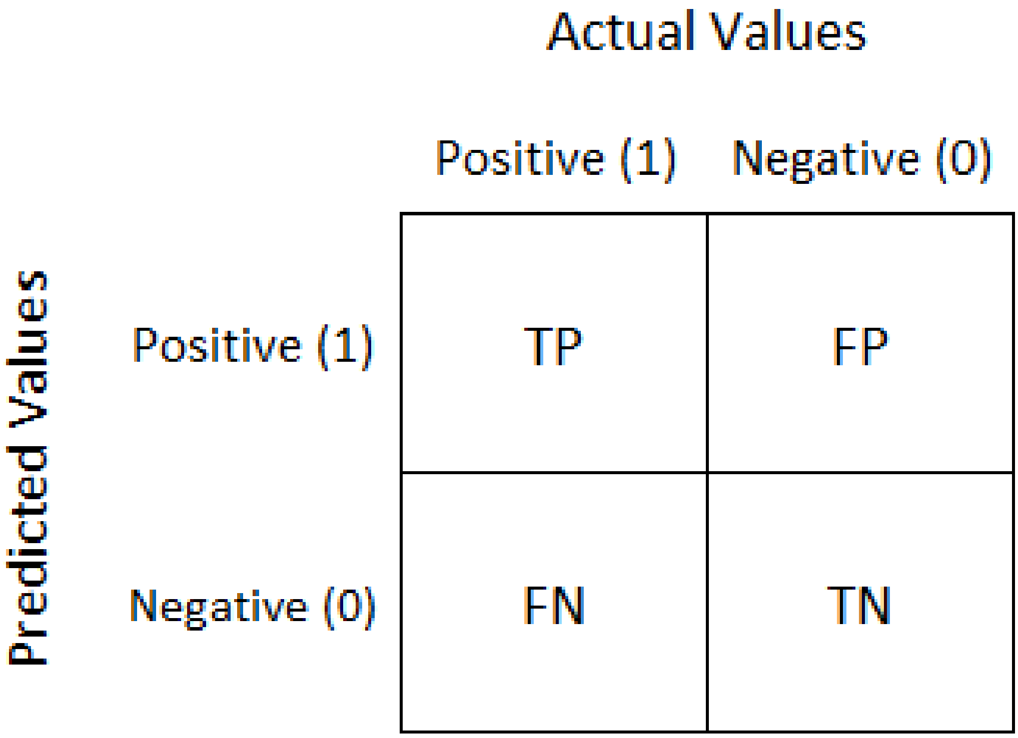



Figure 1.9: Binary classifier confusion matrix [21]

Figure 1.9 shows the confusion matrix of a binary classifier, we define the following terms:

**Predicted Values:** is the values that are predicted by the model.

**Actual Values:** is the values that are actually in a dataset.

**True Positive(TP):** is the values that are actually positive and predicted positive.

**False Positive(FP):** is the values that are actually negative but predicted to positive.

**False Negative(FN):** is the values that are actually positive but predicted to negative.

**True Negative(TN):** is the values that are actually negative and predicted to negative.

Table 1.1 shows the different metrics of machine learning algorithms.

Table 1.1: Classification metrics.

| ***Parameters*** | ***Formula*** |
|---|---|
| True-Positive Rate (TPR)<br>Recall<br>Sensitivity | $\frac{TP}{TP+FN}$ |
| True-Negative Rate (TNR)<br>Specificity | $\frac{TN}{TN+FP}$ |
| False-Positive Rate (FPR)<br>1 - Specificity | $\frac{FP}{FP+TN}$ |
| False-Negative Rate (FNR) | $\frac{FN}{FN+TN}$ |
| Positive Predictive Value (PPV)<br>Precision | $\frac{TP}{TP+FP}$ |
| Accuracy | $\frac{TP+TN}{TP+TN+FP+FN}$ |
| F_Measure<br>F1_score | $\frac{2*Precision*Recall}{Precision+Recall}$ |

## 1.4 Conclusion

In this chapter, we presented Android operating system, its main components and different attacks. Furthermore we introduced machine learning techniques and their metrics.

In the next chapter we explain botnet attacks and their threats to users and servers.

# Chapter 2

# Botnet

## 2.1 Introduction

Cyber attacks have increased greatly, especially in recent time, which led researchers to look for new ways to protect users by detecting and preventing such attacks.

There are multiple types of attacks such as: malware attack, botnet attack, phishing attack, etc.

In this chapter we explain botnet attacks and their threats to users, finally we list some existing android botnets that affected the world greatly.

## 2.2 Definitions

A bot is a compromised host that can be controlled from external servers/devices by a master (*botmaster*) to conduct different malicious attacks such as Distributed Denial of Service(DDOS) attack, spam distribution, private information theft, etc.[24]

A botnet is a network that contains the bots and a Command and Control (C&C) infrastructure that allows the bots to get commands, receive updates, and send their current status information to the botmaster(s), an overview of the botnet architecture is shown in Figure 2.1. [25]

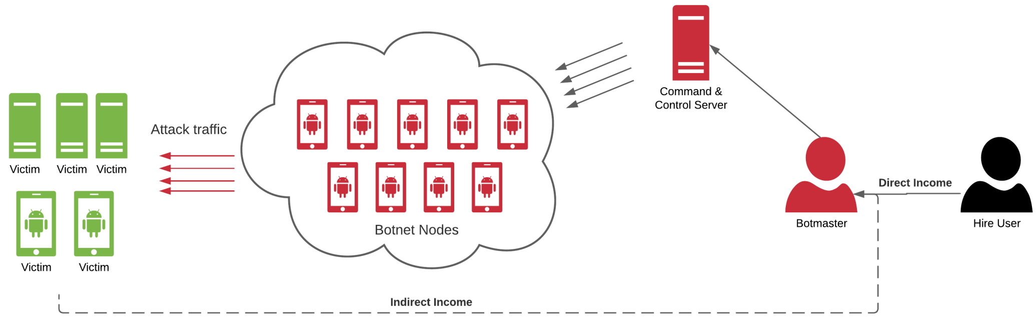
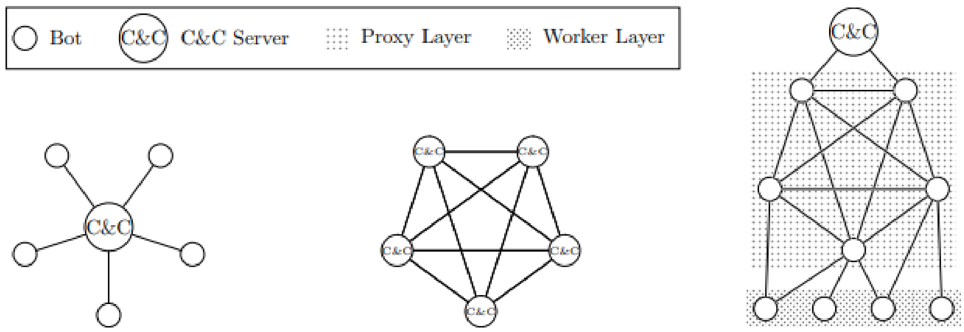



Figure 2.1: Botnet architecture overview.

## 2.3 Topologies

According to [26] there are multiple topologies for botnets and each one has its advantages and disadvantages.




(a) Centralized botnet: One or multiple C&C server(s), multiple bots.

(b) Fully meshed P2P botnet: No dedicated C&C server, every bot can send commands.

(c) Hybrid P2P/C&C botnet: Combines P2P and centralized.

Figure 2.2: Botnet topologies [26]

### 2.3.1 Centralized topology

This topology consists of a dedicated C&C server connected directly to each bot as shown in Figure 2.2-a.

The advantage of this topology is that it is easy to deploy and has low latency also it is highly scalable.

The disadvantage is that it is easy to take down, by shutting down the C&C server(s) the whole botnet will be crippled.

### 2.3.2 Peer To Peer topology

This topology consists of bots connected to each other and each one can act as a C&C server as shown in Figure 2.2-b.

The advantage of this topology is that it is hard to cripple the botnet because taking down some bots doesn't mean taking down the whole botnet.

The disadvantage is that it is hard to implement, hard to add or remove a bot, also fully connected network doesn't scale due to the number of connection that is required for each bot which is limited by operating systems.

### 2.3.3 Hybrid topology

This topology takes advantages of both Centralized and Peer to Peer topologies by combining them into layers to enforce the botnet and avoid some disadvantages of the two topologies as shown in Figure 2.2-c.

The advantage of this topology is that it is hard to take down and it is highly scalable.

The disadvantage is that it is complex and requires to design the botnet into layers to make it hard to take down.

## 2.4 Protocols

Botnets can be categorized depending on the communication protocol used between C&C servers and client bots. [27]

The early generations of botnets use Internet Relay Chat (IRC) protocol where botmaster(s) **push** commands to the bots [28] but this communication is centralized and by banning the IRC C&C server the whole botnet will be down. [29]

Later generations of botnets use Hypertext Transfer Protocol (HTTP) where bots **pull** commands from botmaster(s) periodically to check for new commands [28]. This protocol allows them to hide their traffic in the enormous amount of legitimate web traffic and avoid being detected by basic firewalls, but this communication is also centralized and by banning the C&C web server the whole botnet will be down. [27, 29]

Another generation of botnets appeared in 2004 and 2005 use Peer to Peer (P2P) schemes and protocols and due to its decentralized structure there are no dedicated C&C servers and every node acts as a bot and as a C&C server thus allowing them to continue working properly even if some nodes have been banned but this type of communication has high latency and thus impacting the bots synchronization. [28, 29]

The latest generations of botnets use hybrid infrastructure which allows them to take the benefits and avoid the limitations of centralized and decentralized structures. [29]

## 2.5 Botnet attack

### 2.5.1 Attack steps

Haddadi et al. [30] mentioned in their paper that earlier generations of botnets had a list of commands that were set at the infection time, but current generations of botnets use five stages to create and maintain a botnet which is listed below:

1. **Infection stage** the attacker tries to find the vulnerabilities of a host after infecting it using different exploits (malware applications).
2. **Injection stage** the attacker uses the discovered vulnerability to execute a shellcode that downloads the bot binary and installs it into the infected host.
3. **Connection stage:** the bot binary connects to a C&C channel then tries to infect other devices and waits for further commands from the botmaster.
4. **Attack stage:** the bot binary execute other attacks after receiving commands from the botmaster.

5. **Maintenance stage:** the attacker can issue updates to the bot binary through the C&C channel.

An overview of botnet attack steps is shown in Figure 2.3

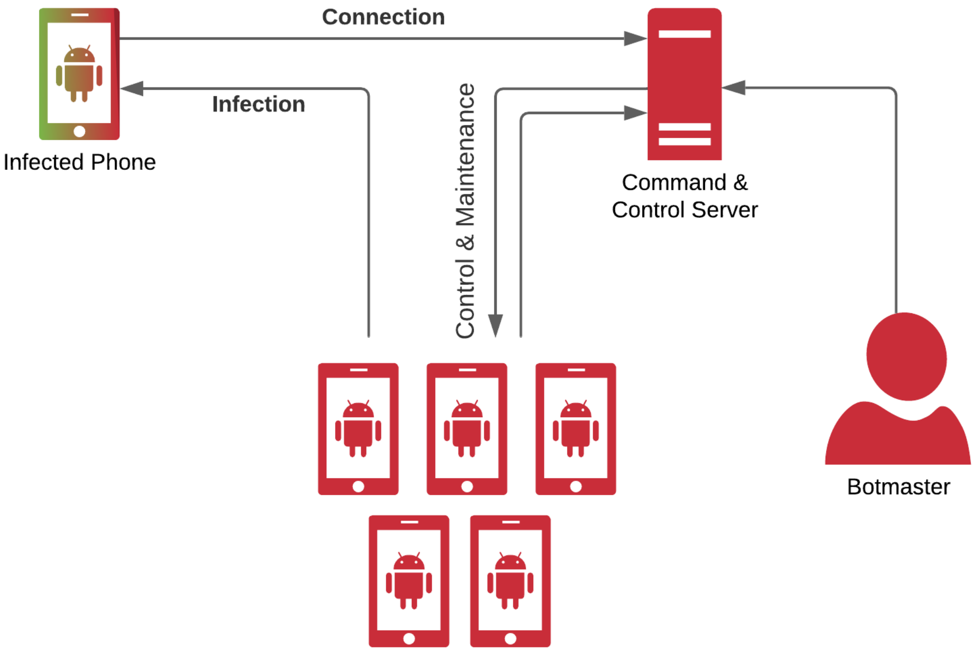



Figure 2.3: Botnet attack steps overview.

### 2.5.2 Attack risks

According to [24, 31] smartphone botnets can be used to launch different attacks such as:

- **Distributed Denial of Service (DDoS):** each bot tries to access the target server at the same time, which makes the server unable to serve all the incoming request including legitimate requests from real clients and by doing so the server owner loses traffic quota and his clients. [32]
- **Phishing:** the bot tries to lure the device owner into providing his personal information such as credit card details and passwords by redirecting him to fake websites which are owned by the botmaster. [33]

- **Click fraud:** the bot shows to the device owner or clicks automatically on pay-per-click advertisements to exhaust the advertiser budget or to make a profit on websites that are owned by the botmaster. [34]
- **Generation and distribution of spam:** the bot can generate spam emails and SMS messages, then send them to other peoples while avoiding the email being marked as spam by email servers, due to the fact that such emails are sent from legitimate devices.
- **Cryptojacking:** the bot can use the host resources such as CPU, RAM, disk space to mine cryptocurrency and generate revenue for the botmaster. [35]
- **Brute-force:** the bot can use the host to brute force passwords on external servers.

## 2.6 Existing botnets

### 2.6.1 Matryosh - 2021

According to [37, 38] 360 netlab BotMon system detected a new botnet on January 25, 2021 that reused Mirai framework, the new botnet targeted Android devices and it propagates through the Android Device Bridge (ADB) interface.

Figure 2.4: Matryoshka: russian nesting dolls [36]

The new botnet is named *Matryosh* because the encryption algorithm which is implemented in it and the process of obtaining C&C are nested in layers like Russian nesting dolls see Figure 2.4.

Matryosh supports many CPU architectures and its main functionality is launching DDoS attacks via TCP, ICMP, and UDP protocols.

After infecting a device Matryosh follows the following steps:

1. It changes the name of its process.
2. It prints "pipe failed" on the *stdin* to confuse Log-based botnet detection methods.
3. It sends a DNS TXT request to the remote hostname to obtain a TOR C&C and a TOR proxy.
4. It establishes a connection with the TOR proxy.
5. It communicates with the TOR C&C through the TOR proxy.
6. It waits for the commands that are sent by C&C to execute them.

### 2.6.2 Chamois - 2016

According to [39, 40] Chamois malware appeared on Google Play in August 2016, in March 2018 Chamois had already infected 20.8 million devices but the current Google Play security measures reduced the number of devices in the botnet by 91%, despite that researchers found 12,800 new samples just between March 2018 and March 2019.

Chamois developers created benign apps that contain Chamois malware to trick Google Play users into installing them, but Google play's app checking tools evolved and started blocking Chamois malware, in response later versions of Chamois mislead app developers and phone manufacturers to incorporate the code directly into their apps thinking that Chamois is a mobile payment solution while developers thought Chamois is an advertising software development kit thus these tainted apps started to appear on Google Play.

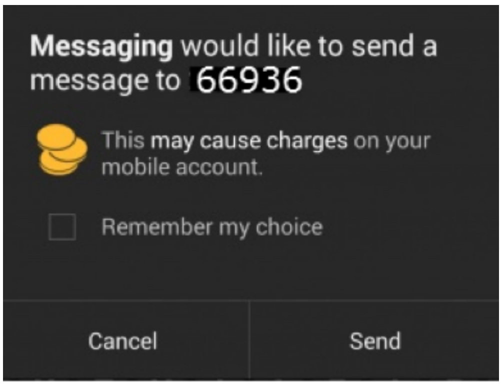



Figure 2.5: Premium SMS warning [41]

The Chamois botnet served malicious advertisements and directed phone owners to premium SMS scams.

The Android security team required apps to obtain explicit permission to text a premium number to prevent premium SMS fraud, however Chamois developers added a check to see if the device was rooted. If it was, the malware used the root privileges to disable premium SMS warnings as shown in Figure 2.5, if it was disabled they used the Accessibility service to automatically click the Send button As a result, the phone owners learned about those messages only after their bills arrive.

Recent versions of Chamois checks if the device contains antivirus, anti-debugging, or anti-analysis tools if it is the malware doesn't execute the malicious code. the botnet included also a mechanism called feature flags, which is used in software development to enable and disable particular features in different parts of the world, Chamois developers use feature flags to test updates to confirm that the updated version is working as expected before pushing the update globally.

Google currently uses several detection methods to identify Chamois, including signature-based flags, machine-learning assessment, and behavioral analytics. Google also uses Google Play Protect app to scan pre-installed apps to check for situations where Chamois is incorporated in a legitimate package, Google also encourages phone manufacturers to audit third-party code before shipping it on to their phones.

### 2.6.3 WireX - 2017

According to Kaspersky Lab's DDoS Intelligence Report for the third quarter of 2017 [42] WireX a botnet with several hundred thousand bots at its peak, was taken down.

WireX had been working undercover on Android devices and replicating through legitimate Google Play applications. WireX perform volumetric DDoS attacks which can overwhelm DDoS mitigation systems by the high volumes of the malicious traffic.

The WireX indicators were first available on August 2nd 2017 as minor attacks that went unnoticed at the time, until the researchers began looking for the 26-character user-agent string in the logs. These initial attacks indicated that the malware was under development and more prolonged attacks were identified starting August 15 2017, as shown in Figure 2.6.

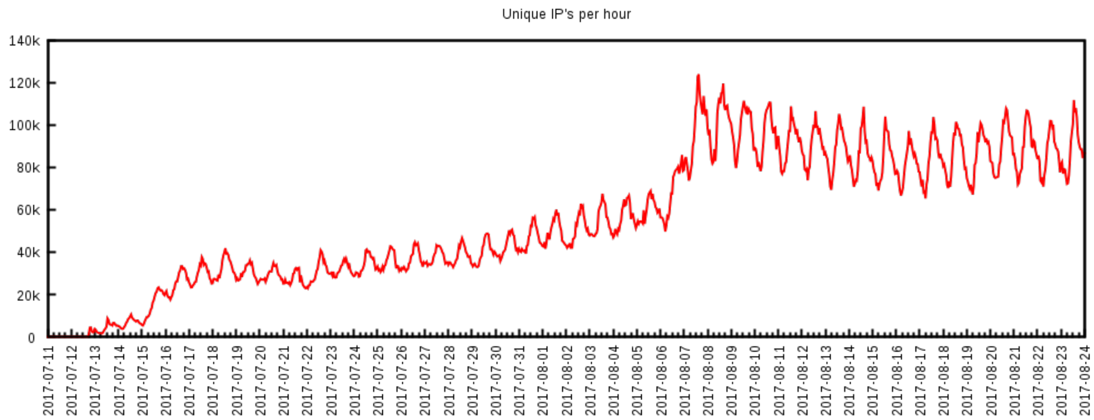



Figure 2.6: WireX botnet growth [43]

### 2.6.4 Geost - 2016

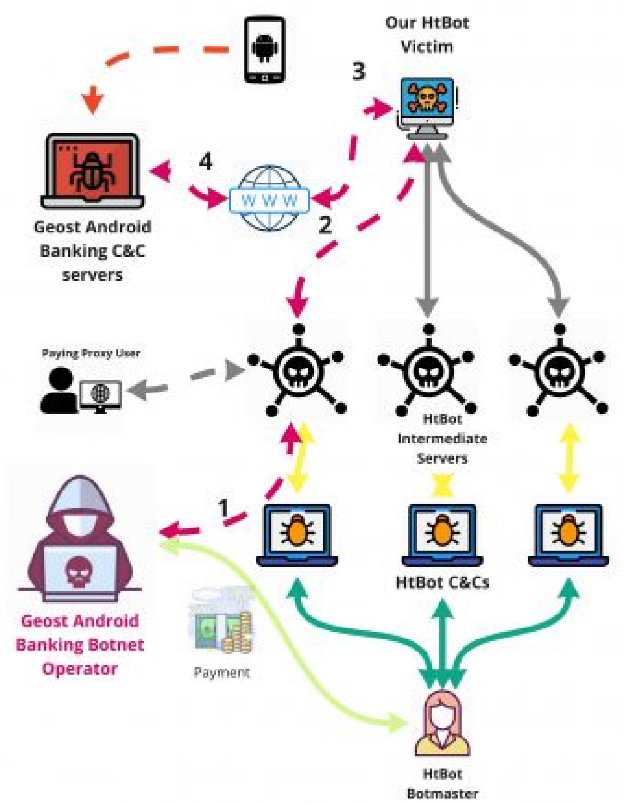



Figure 2.7: Geost botnet: attack steps [44]

According to [45] the Geost botnet has been in operation since at least 2016 and it consists of at least 140 (C&C) servers, 140 domains, more than 140 Android packages, more than 800k infected Android devices. The botmasters of Geost botnet are researchers from the Czech Technical University with researchers from UNCUYO university.

Geost botnet targets online banking users, it was mainly focused on five banks in Eastern Europe and Russia. the attack of the Geost botnet steps are shown in Figure 2.7.

Geost botnet was discovered after using HtBot to manage infected hosts without knowing that Avast themselves created HtBot.

## 2.7 Conclusion

Attackers preform botnet attacks on Android devices due to the never ending increase in the smartphone market.

In the next chapter we present different types of Android applications analysis, furthermore we list the latest Android botnet detection methods.

# Chapter 3

# State of the art

## 3.1 Introduction

Botnet attack allows attackers to manipulate and control user's devices through a Command and control (C&C) server to launch other attacks on behalf of the attacker.

There are different methods for analyzing Android applications, which are: static, dynamic, and hybrid analysis.

In this chapter we present the latest techniques used to detect Android botnets.

## 3.2 Static analysis

According to [46], in static analysis, the application's apk file is analyzed without executing it and some go beyond that by reverse engineering the apk file to extract the source code also.

The advantages of this method are that it can identify suspicious code that only executes under specific conditions and also this method uses less resources than other methods.

The disadvantage is that it can't detect encrypted content or any downloaded content from external servers.

### 3.2.1 Detection using Convolutional Neural Networks

Hojjatinia et al. [47] proposed a new method based on Android permissions using convolutional neural networks (CNN) to classify botnets and benign Android applications. They also proposed a new method to represent each application as an image created based on the co-existence of the permissions used in that application.

The researchers use the ISCX dataset [48] then they selected 1,800 Android Botnet samples from 14 different families.

To collect benign samples, they developed a tool to crawl the Google Play store then they downloaded 3650 samples from 24 different categories. All benign samples have been scanned using VirusTotal [49] to ensure that the benign category doesn't include any malware sample.

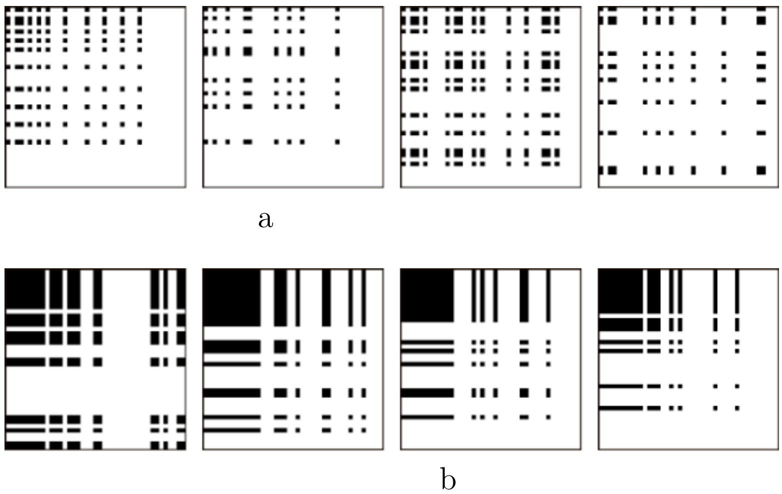



Figure 3.1: Hojjatinia et al. [47] image representation of (a) benign and (b) botnet android applications.

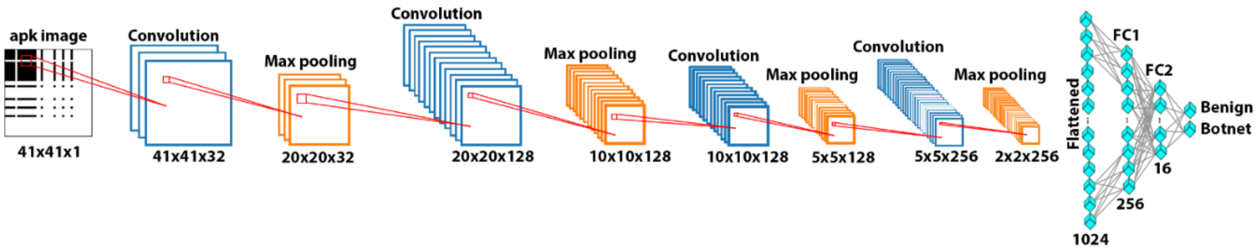



Figure 3.2: Hojjatinia et al. [47] CNN model architecture

| Layer | Input tensor size | Type | Activation Function | Kernel size | Strides | Kernels | Output tensor size |
|---|---|---|---|---|---|---|---|
| 1 | $(41,41,1)$ | Conv | ReLU | $5\times5$ | $(1,1)$ | 32 | $(41,41,32)$ |
| 2 | $(41,41,32)$ | Max-pool | - | $2\times2$ | $(2,2)$ | - | $(20,20,32)$ |
| 3 | $(20,20,32)$ | Conv | ReLU | $5\times5$ | $(1,1)$ | 128 | $(20,20,128)$ |
| 4 | $(20,20,128)$ | Max-pool | - | $2\times2$ | $(2,2)$ | - | $(10,10,128)$ |
| 5 | $(10,10,128)$ | Conv | ReLU | $3\times3$ | $(1,1)$ | 128 | $(10,10,128)$ |
| 6 | $(10,10,128)$ | Max-pool | - | $2\times2$ | $(2,2)$ | - | $(5,5,128)$ |
| 7 | $(5,5,128)$ | Conv | ReLU | $1\times1$ | $(1,1)$ | 256 | $(5,5,128)$ |
| 8 | $(5,5,256)$ | Max-pool | - | $2\times2$ | $(2,2)$ | - | $(2,2,256)$ |
| 9 | $(2,2,256)$ | FC-1 | ReLU | - | - | - | $(256,1)$ |
| 10 | $(256,1)$ | FC-2 | ReLU | - | - | - | $(16,1)$ |
| 11 | $(16,1)$ | Softmax | - | - | - | - | $(2,1)$ |

Figure 3.3: Hojjatinia et al. [47] The proposed CNN model configuration

The researchers extracted the permissions of both botnet and benign applications into two different lists sorted by the frequency of the permissions, then they merged the two lists into one list sorted by the frequency of the permissions, next they selected the top 41 frequently used permissions from the merged list. The image representation of each app is a matrix of 41 x 41 where the [i, j] element shows the co-occurrence of the $i_{th}$ and the $j_{th}$ permissions in the application which mean if both permissions are used by the application, the [i, j] element is set to 0, otherwise, it is set to 255. The figure 3.1 shows some samples of images created for botnets and benign applications.

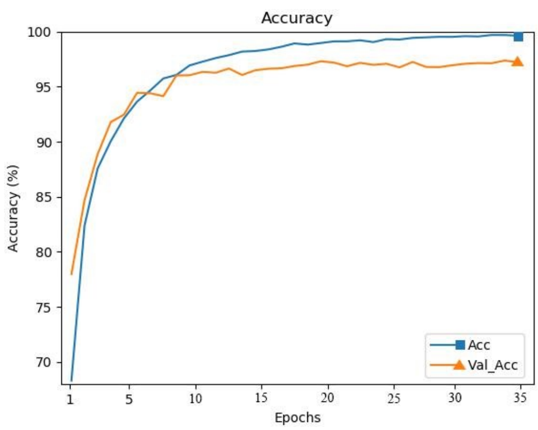



Figure 3.4: Hojjatinia et al. [47] CNN model accuracy by epochs

The researchers trained a CNN model to distinguish Botnet applications from benign ones using the image representation of the applications. The figure 3.2 illustrates the architecture of the CNN model.

The researchers trained and tested the proposed CNN model in 10, 15, 20, 25, 30, and 35 epochs as shown in figure 3.4. The researchers used 10-fold cross validation to evaluate their proposed method and the results indicate that their proposed method is quite successful in classifying benign and botnet applications with an accuracy of 97.2%.

The researchers achieved an accuracy of 97.2% using only Android permissions which is pretty impressive.

### 3.2.2 Detection using Random forest

Anwar et al. [50] proposed a new framework to detect botnet applications using static analysis.

Initially, the researchers obtained 1865 benign applications from Google Play, then they used VirusTotal tool [49] to confirm their cleanness, next they used the Monte Carlo method [51] to remove duplicated applications, as a result they obtained in total 1330 benign applications.

The researchers collected botnet applications from different datsets such as DREBIN [52], ISCX Android Botnet Dataset [48], Android Malware Genome Project [53] and to simplify machine-learning modelling they obtained only 1330 botnet applications to match the total number of benign applications.

The researchers proposed a framework that has five layers as shown in figure 3.5 which are the decompiler, extractor, smart learner, features refiner, and the machine learning module.

1. **App Decompiler:** is responsible for converting the apk file into a readable format which is used for further analysis, in their study they used the Android asset packaging tool (AAPT) for this task.
2. **Feature Extractor:** is responsible for generating a CSV file for each decompiled app. The CSV file contains the extracted features after reverse engineering the app using the Androguard open-source tool [54], the extracted features are permissions, activities, broadcast receivers, services, and API calls. The researchers found out that botnet applications usually use more features, permissions and API calls than benign applications.
3. **Smart Learner:** is responsible for generating feature patterns from the generated CSV files using the Apriori algorithm in the WEKA tool [55]. The researchers used the Apriori algorithm to extract significant features combination after indexing all the extracted features.
4. **Feature Refiner:** is responsible for selecting the most related features to botnet applications, in their study they used information gain (IG) algorithm on the botnet applications dataset to rank application features, then they selected only the high ranked features to be used in the machine learning model.

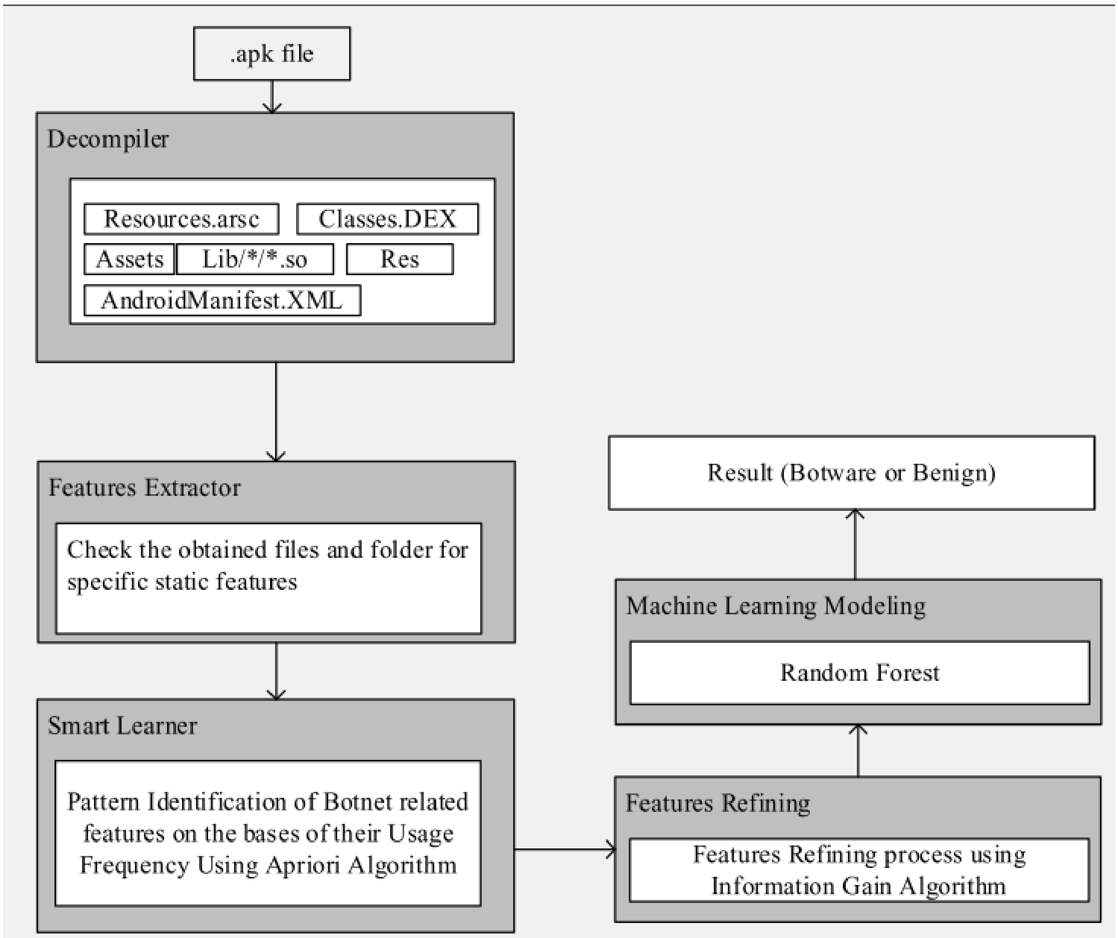



Figure 3.5: Anwar et al. [50] method diagram

5. **Machine Learning Modelling:** is responsible for training the Machine Learning classifier, for this study they chose support vector machine (SVM), Random Forest, J-48, simple logistic regression (SLR), and Naïve Baye algorithms to test their framework. As an input, they used the selected features from the Feature Refiner stage.

The performance of the proposed framework was evaluated using the following metrics True Positive Rate (TPR), False Positive Rate (FPR), Precision, F-measure, and the Accuracy metric.

The researchers conducted the experiment separately on the permissions, the activities, the broadcast receivers, the services, and the API calls features then on the combined features set.

The researchers found that random forest algorithm and using the combined features set has the highest accuracy of 0.9820, while TPR is 0.7880, precision is 0.8893 and FPR is 0.1140 but, the experiment produced also a low F-measure of 0.7457.

The researchers proposed a framework that can detect botnet application with an accuracy of 98.2% using random forest algorithm because it can ignore unrelated features to Android botnet attacks however it can be improved by introducing dynamic features.

## 3.3 Dynamic analysis

According to [46, 56, 57], in dynamic analysis, the application is analyzed while executing it by monitoring network traffic, system logs, etc.

The advantage is that it can detect encrypted content or any downloaded content from external servers and it provides better accuracy over pure static analysis methods.

The disadvantages are that it requires a lot of resources to emulate a full Android system and also it can't detect any suspicious code that only executes under specific conditions that weren't met while executing the application, also there are applications that they can detect that they are being monitored or are running under an emulated device.

### 3.3.1 Detection using PSO-SVM

Moodi et al. [17] proposed a dynamic approach to detect Android botnet applications using Smart Self-Adaptive Learning-based PSO-SVM (SSLPSO-SVM) method.

The researchers used the 28 standard Android botnet dataset [58], which is created after collecting 14 million packets of network traffic and contains 85 different features from 336,111 application, 189,842 of the applications are benign (59.57%) and 146.269 are botnet applications (40.43%).

The researchers used SVM with radial basis function (RBF) kernel for classification, while RBF requires a parameter $\sigma$ that has to be set. SVM has a parameter

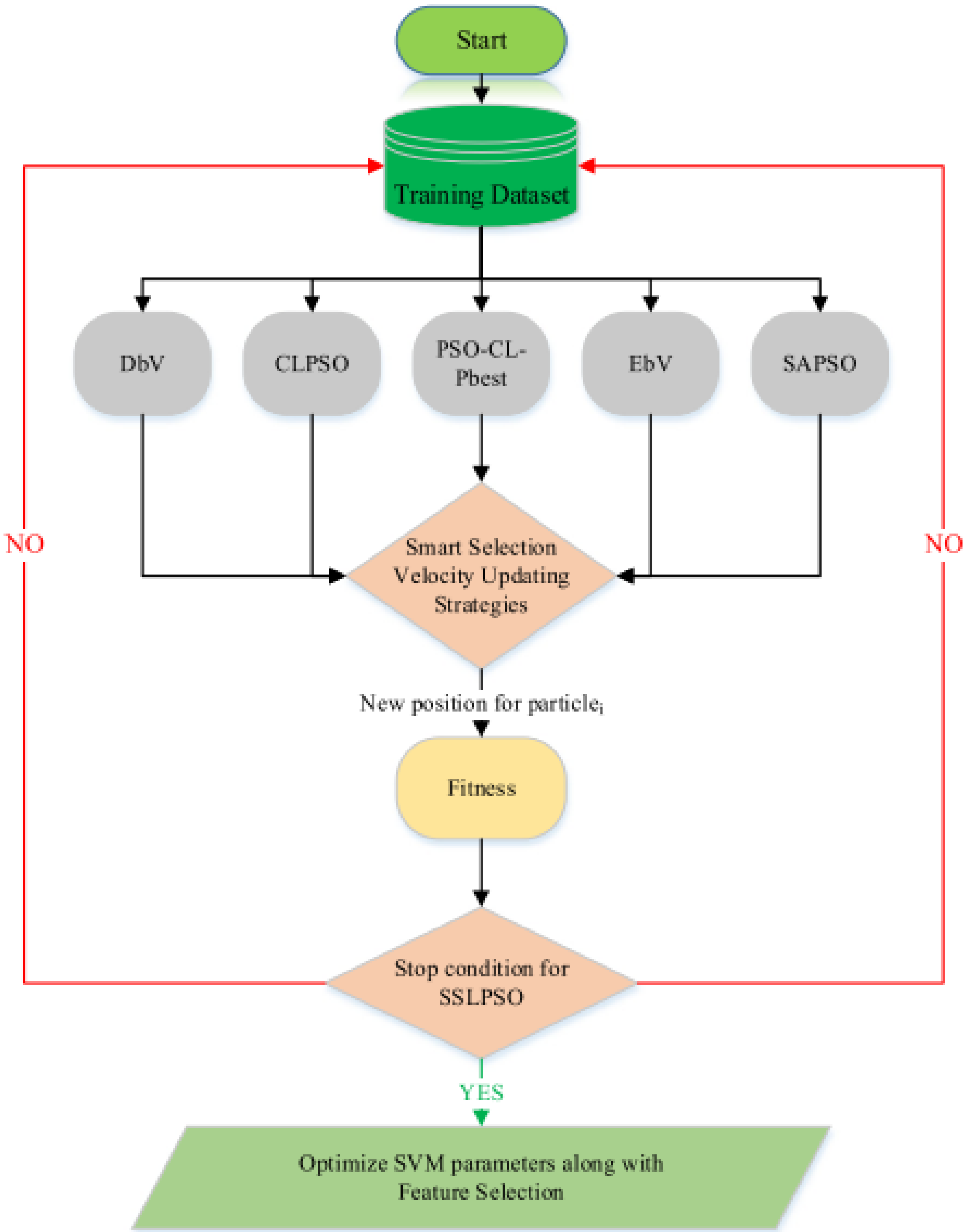



Figure 3.6: Moodi et al. [17] method overview

called Penalty (C) and its value has to be set also; however, to obtain the best results the researchers used their method SSLPSO-SVM to find the best value of the parameters $\sigma$ and Penalty (C) to achieve accurate results.

The input of SVM is the application features and for optimal results in less time, only the important features must be selected, for that the researchers used Binary PSO (BPSO) method [59]. In BPSO each particle in feature selection has one of two states either Selecting the feature (1) or Lack of selecting the feature (0).

SSLPSO method uses five different algorithms to update the particle velocity as shown in Figure 3.6. The algorithms that SSLPSO method uses are introduced below:

- **Difference-based Velocity (DBV):** [60] this method can avoid sudden changes in velocity by updating the particle velocity based on multiple information from the search space which results in particles looking for a larger space to update their velocity.
- **Comprehensive Learning PSO (CLPSO):** [61] this method can update the velocity on multi-modal issues by allowing each particle to affect other particles Pbest.
- **PSO-CL-Pbest:** [62] CLPSO method has a low convergence velocity, to fix that this method reduces the algorithm complexity by using a random function to select Pbest for all particles in all dimensions.
- **Estimation-based Velocity (EbV):** [62] the PSO main algorithm has a high convergence rate however after a few steps the particles lose their efficiency by getting trapped in local optimal points, to fix that this method use speed upgrades for complex multi-modal issues which allows it to have a high convergence velocity.
- **Smart Adaptive PSO (SAPSO):** [63] this method selects the core parameters (, c1 and c2) of the velocity formula dynamically, if we assume that particles can experience during the execution.

While Smart Adaptive PSO (SAPSO) [63] method uses Roulette Wheel Selection(RWS) method for algorithm selection which relies on a random function making it possible that the best algorithm may not be selected in every iteration. The SSLPSO method uses an approach called Smart Selection Strategies(SSS) which selects from the five algorithms the best preferment algorithm(s) and it provides them with more particle while less preferment algorithm(s) get less particles for the next iteration, the best preferment algorithm(s) are the ones that made the largest number of changes in Particle best (Pbest) and Global best (Gbest).

The researchers compared their method SSLPSO against three other methods: SLPSO, CLPSO and PSO-CL-Pbest. for fair results they assumed that the three methods can optimize SVM parameters while selecting the important features.

For the experiment the researchers studied the effect of the data volume and the balance of the data, as a result they found that SSLPSO preformed better or equally to some other algorithms in every scenario.

The researchers found that SSLPSO method has the highest Sensitivity, Specificity, Precision and Accuracy while being the most time-efficient amongst the other three algorithms.

The researchers claims that in average SSLPSO achieves the Accuracy of 98.2829%, the Precision of 97.7386%, the Specificity of 95.5300%, the Sensitivity of 96.7604%.

SSLPSO method can achieve great results because it optimizes SVM parameters ($\sigma$ and C) while selecting only the important features.

### 3.3.2 Detection using Random Forest

da Costa et al. [64] presented an anomaly-based and host-based approach for detecting mobile botnets. The proposed approach uses machine learning algorithms to identify anomalous behaviors in statistical features extracted from system calls.

To extract system calls the researchers rooted their Samsung Galaxy tablet with Android 4.1.2, then they installed the Strace tool [65] on it. The tablet is connected via USB to a Windows 10 laptop and is also connected via WiFi to a network hosted by the computer, which has Internet access.

The researchers collected botnet applications from ICSX Android Botnet dataset, which contains different families of botnets. They chose 31 botnet applications, divided into 13 different families. For legitimate apps the researchers installed apps from Google Play directly into their tablet.

The researchers proposed a system that consists of three parts as shown in Figure 3.7:

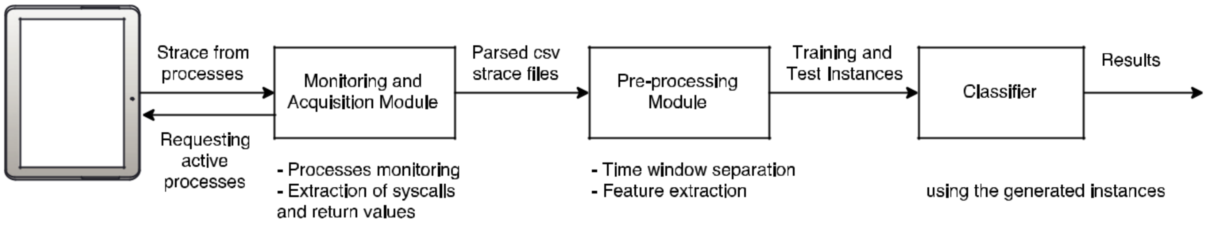



Figure 3.7: da Costa et al. [64] method overview

1. **Monitoring and Acquisition module:** this module is used to collect the data needed in the other modules.
2. **Pre-processing module:** this module is used to extract information from the data and creating instances of classifiers.
3. **Classifier module:** this module is used to classify benign and botnet applications activity.

Using the Monitor and Acquisition module, The researchers collected data from the device while only legitimate applications were running. Then, they introduced 31 botnet applications one by one, next they sampled a random set of botnet applications and installed them together in the device.

The researchers analyzed the Strace files generated by the Monitoring and Acquisition module file to extracting system calls, then the pre-processor module grouped the system calls by different time windows, (500ms, 1s, 5s, and 10 s), next, they created a feature vector for each time window.

In the classifier module, The researchers used two machine learning algorithms, namely Random Forest and SVM with linear kernels and SVM with RBF kernels.

The researchers used 60% of the dataset for the training process and the rest, 40%, was used for testing. The experiment was repeated 50 times, taking several random samples from the dataset.

The researchers compared the performance of the different machine learning algorithms using the receiver operating characteristic (ROC) curve which is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

The researchers found from the plotted ROC curve in Figure 3.8 that random forest algorithm achieved better results compared to the other machine learning algorithms.

The researchers achieved a precision of 0.866 within a 500ms time window using random forest algorithm.

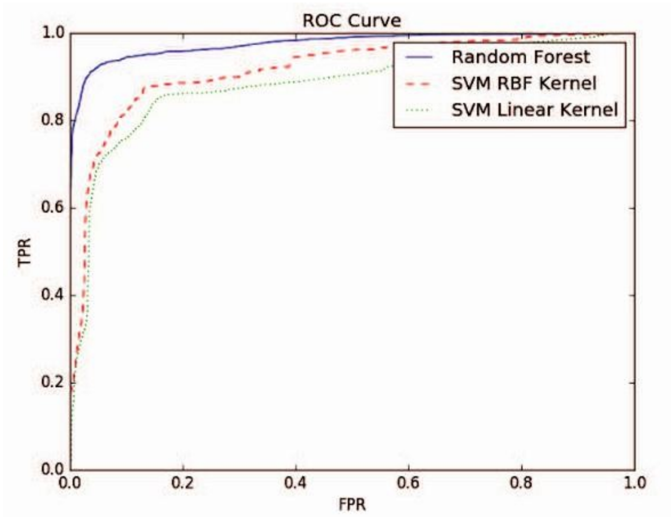



Figure 3.8: da Costa et al. [64] method ROC curve

## 3.4 Hybrid analysis

In hybrid analysis, we try to eliminate some limitations in static and dynamic analysis by combining them together.

The advantage of this method is that we can take the advantages of static and dynamic analysis to achieve a high accuracy.

The disadvantages are that it requires more resources and thus it is hard to train on a big dataset. [46]

Karim et al. [66] Proposed a proof of concept framework to detect botnet applications using hybrid analysis, their purpose is to prove that hybrid analysis is more effective for detecting botnet applications.

The researchers used two datasets:

1. **Evaluation dataset:** this dataset is used to study and analyze benign, malware, botnet applications and their related features. this dataset contains 10 application from each category (benign, malware, botnet).
2. **Validation dataset:** this dataset is used to validate their classifier model. The dataset contains 1371 botnet applications collected from ICSX dataset [48] and 500 benign application collected from Androtracker dataset. [67]

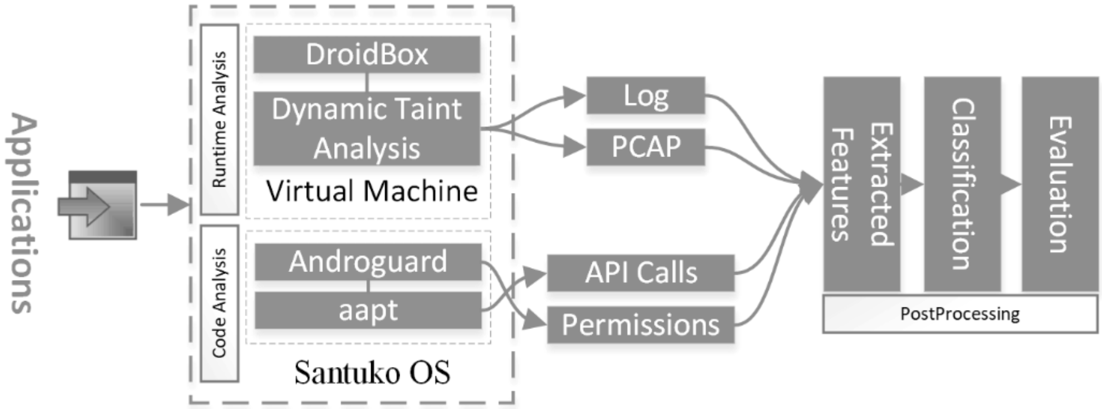



Figure 3.9: Karim et al. [66] method overview

The researchers used Androguard tool [54] for static analysis and they extracted the following features: permissions and API calls.

The researchers used DroidBox tool [68] for dynamic analysis and they extracted the following features: file activities, network operations, Information Leaks, Services, SMS operations, Cryptographic Operations, DNS Traffic, HTTP Traffic, unknown Conversations.

The researchers split their study in two steps:

1. **Analysis step:** in this step the researchers used the evaluation dataset and extracted the static and dynamic features of each app then they compared the results of each application category (botnet, malware, benign) to extract only the important features that is used by most botnet applications, after that they used multiple classifiers such as J48, Naïve Bayes, Random Forest, and Logistic Regression to demonstrate that they can classify botnets using the extracted important features.

2. **Validation Step:** in this step the researchers used the Validation dataset to demonstrate that their model using the selected features can detect Android botnet on a large dataset, for this they used multiple classifiers such as J48, Naïve Bayes, Random Forest, and Logistic Regression.

The researchers used multiple classifiers such as J48, Naïve Bayes, Random Forest, and Logistic Regression with static only, dynamic only, hybrid features to prove that hybrid analysis is the best way to classify Android botnet applications.

In the analysis step and using the evaluation dataset the researchers found that Random Forest classifier performed better than other classifiers with an accuracy of 90% using the hybrid analysis.

In the validation step and using the validation dataset the researchers found that Random Forest classifier performed better than other classifiers with an accuracy of 98% using the hybrid analysis.

Yusof et al. [69] tried a similar method to Karim et al. [66] method, however Yusof et al. [69] collected for their experiment 1,527 botnet application from Drebin dataset [52] and they downloaded 800 application from Google Play then tested them using VirusTotal tool [49] to verify their cleanliness before using them as benign application after that they extracted permissions, API calls, and system calls as features from each application in the their dataset as a result they achieved an accuracy of 97.9% using Random Forest algorithm which their best classifier among other classifiers.

The researchers tried to prove that hybrid analysis is the best way to classify Android botnet applications however looking at their experiment results we can see that their best classifier which is Random Forest has an accuracy of 98% using only static features which is the same when using hybrid features.

.

Table 3.1: Application analysis types comparison.

| Analysis Type | Advantages | Disadvantages |
|---|---|---|
| **Static Analysis** | • It can identify suspicious code that only executes under specific conditions.<br>• It uses less resources than other methods. | It can't detect encrypted content or any downloaded content from external servers. |
| **Dynamic Analysis** | • It can detect encrypted content or any downloaded content from external servers.<br>• It provides better accuracy over pure static analysis. | • It requires a lot of resources to emulate a full Android system.<br>• It can't detect any suspicious code that only executes under specific conditions that weren't met while executing the application.<br>• There are applications that can detect that they are being monitored or are running under an emulator device. |
| **Hybrid Analysis** | It takes the advantages of static and dynamic analysis to achieve the best accuracy. | It requires more resources and thus it is hard to use a big dataset for testing and training. |

## 3.5 Conclusion

There are different methods to detect Android botnet applications and each one has it's advantages and disadvantages as shown in Table 3.1, therefore it is important to find a model that can detect Android botnet applications with a high accuracy.

In the next chapter we propose a model that can detect Android botnet applications efficiently.

# Chapter 4

# Conception & Implementation

## 4.1 Introduction

Android botnet detection methods are a major topic in the smartphone security field, and we need methods that achieve high accuracy to protect smartphone users against botnet attack.

In this chapter we propose a model that depends on static analysis to detect Android botnet applications, furthermore we introduce the dateset we use in our experiment, the environment, the architecture. finally we discuss the obtained results and compare our models against similar models from other methods.

## 4.2 Dataset

After searching for an Android botnet datasets that contains only Android botnet applications we found the ICSX dataset [48] which contains 1929 botnet application from 14 different families which are listed in Table 4.1.

The ICSX dataset contains only botnet application however to train a classifier we need also benign applications and the most used way in other researches to get them is download application from Google Play Store then scan each application using VirusTotal [49].

Table 4.1: Dataset families.

| Family | Year of discovery | No. of samples |
|---|---|---|
| AnserverBot | 2011 | 244 |
| Bmaster | 2012 | 6 |
| DroidDream | 2011 | 363 |
| Geinimi | 2010 | 264 |
| MisoSMS | 2013 | 100 |
| NickySpy | 2011 | 199 |
| Not Compatible | 2014 | 76 |
| PJapps | 2011 | 244 |
| Pletor | 2014 | 85 |
| RootSmart | 2012 | 28 |
| Sandroid | 2014 | 44 |
| TigerBot | 2012 | 96 |
| Wroba | 2014 | 100 |
| Zitmo | 2010 | 80 |

In our study we use static analysis due to time limitations and lack of resources because dynamic analysis is time consuming and requires powerful CPUs and too much RAM to emulate full Android system.

Yerima and Alzaylaee [25] collected 1929 botnet application from ICSX dataset [48], and 4873 benign application from Google Play for 24 categories after that the researchers scanned benign applications using VirusTotal [49] to ensure their safety, and after preforming a static analysis on each application they extracted 342 static features with five different types as shown in table 4.2, the most important feature are shown in table 4.3, all the feature are listed in the appendix 4.6.

Table 4.2: Dataset feature types.

| Feature type | Number |
|---|---|
| API calls | 135 |
| Permissions | 130 |
| Commands | 19 |
| Extra files | 5 |
| Intents | 53 |
| **Total** | **342 features** |

In our study we use Yerima [70] dataset because it has the largest amount of features, it was a result of studying of the whole ICSX Android botnet dataset [48], it contains a very large amount of benign applications, and also it allows us to compare our model directly with the powerful CNN model of Yerima and Alzaylaee [25].

Table 4.3: Dataset most important features.

| **Feature name** | **Type** |
|---|---|
| TelephonyManager.*getDeviceId | API |
| TelephonyManager.*getSubscriberId | |
| abortBroadcast | |
| Ljava.net.InetSocketAddress | |
| io.File.*delete( | |
| System.*LoadLibrary | |
| SEND_SMS | Permission |
| DELETE_PACKAGES | |
| PHONE_STATE | |
| SMS_RECIVED | |
| READ_SMS | |
| ACCESS_FINE_LOCATION | |
| INSTALL_PACKAGES | |
| CAMERA | |
| Android.intent.action.BOOT_COMPLETED | Intent |
| android.intent.action.POWER_CONNECTED | |
| android.intent.action.BATTERY_LOW | |
| chown | Command |
| chmod | |
| Mount | |
| .apk | Extra File |
| .zip | |
| .dex | |
| .jar | |
| .so | |

## 4.3 Environment

We implemented our models in Google Collab without any GPU acceleration and using an Intel Xeon CPU @ 2.30GHz with 13GB of RAM.
Google Colaboratory [71] offers a free Environment that contains pre-installed

libraries for machine learning and data analysis, it also allows the users to write and execute Python code directly from the browser.

To implement our model, we use:

- **Python v3.7.10:** it is a high-level programming language and it is heavily used in artificial intelligence field because it has an active community, a huge library set and also it easy to learn. [72]
- **TensorFlow v2.5.0:** it is a open source library developed by Google to make machine learning and deep learning easier and approachable. [11]
- **Keras v2.5.0:** it is an open source library for Python and it offers an interface to machine learning backends such as TensorFlow and it helps developers write easy to understand and maintainable code. [73]
- **Scikit-learn v0.22.2:** it is an open source library for Python and it offers a set of helpful methods to deal with data also it can perform various machine learning algorithms. [74]
- **Pandas v1.1.5:** it is an open source library for Python and it offers a set of helpful methods for data manipulation and analysis. [75]

## 4.4 Network architecture

Most researchers in the field Android botnet detection relied on SVM, Random Forest, CNN however we tried different solutions using Perceptron neural networks.

Perceptron neural network are introduced by Rosenblat [76] after being inspired by biological neurons and their ability to learn, the original Preceptron uses the Heaviside step activation function and it consists of one input layer and one output node as shown in Figure 4.1, due to that it can classify only linearly separable data, later on Multi-layer Preceptron (MLP) was introduced to solve the problem of the original Preceptron of classifying only linearly separable data.

Multi-layer Preceptron consists of one input layer and $n$ hidden layers and one output layer and uses non-linear activation functions, MLP are considered to be one of deep learning models because it can have many hidden layers.

In our comparative study we implement four different models, all models have an input layer that consist of 342 node and the output layer consists of one node

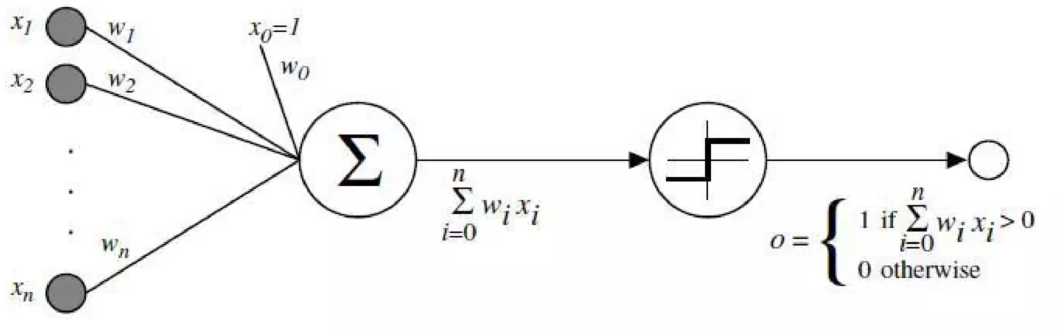



Figure 4.1: A diagram showing how the Perceptron works [77]

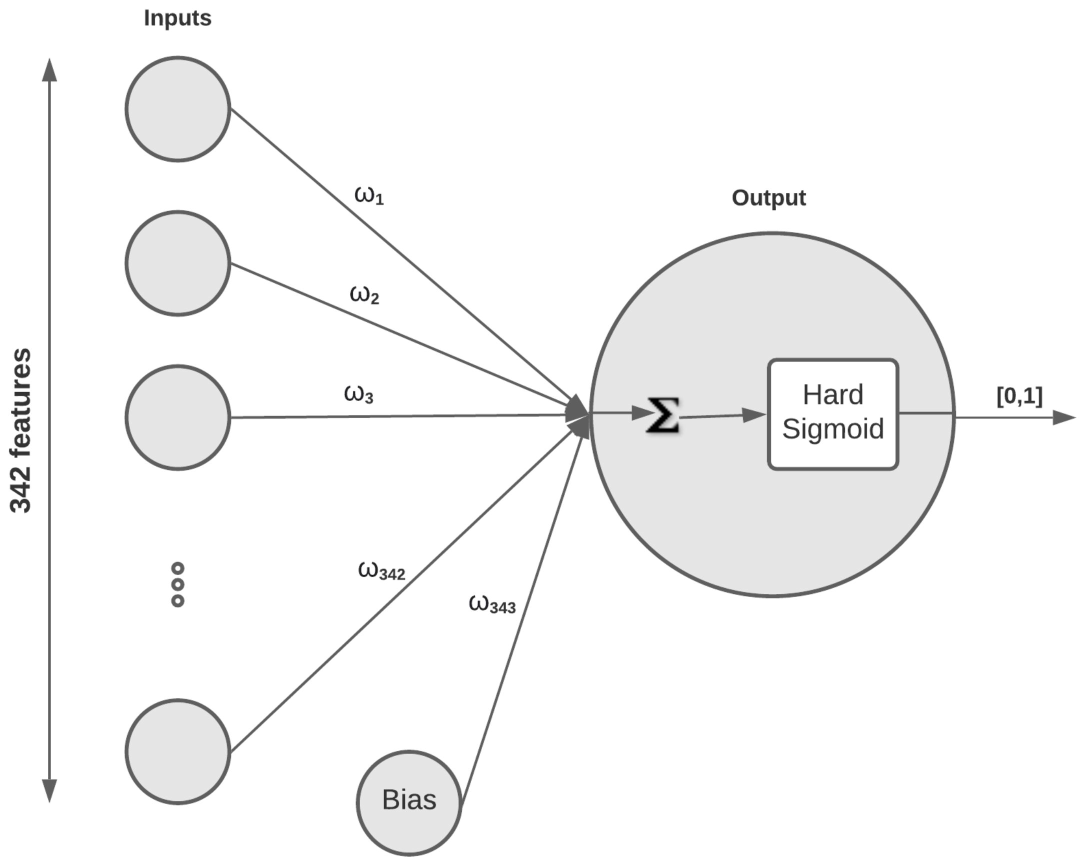



Figure 4.2: Proposed method using the Preceptron model diagram

which returns the probability of an application is a botnet as a value between 0 and 1. the different implemented models are listed below:

1. **Preceptron:** this model has no hidden layers as shown in Figure 4.2 and the model summery is shown in Table 4.4, we use the hard sigmoid activation function from TensorFlow because Keras doesn't support the Heaviside step activation and also it allows our model to classify non linearly separable data. the hard sigmoid activation is defined in TensorFlow as follows:

$$f(x) = \begin{cases} 0, & \text{if } x < -2.5 \\ 1, & x > 2.5 \\ 0.2x + 0.5, & \text{otherwise} \end{cases}$$

2. **MLP-1 hidden layer:** this model has one hidden layer that consist of 170 node, and it uses the ReLU activation function, the output node uses the sigmoid activation function. The model summery is shown in Table 4.5.

3. **MLP-2 hidden layers:** this model uses the ReLU activation function in hidden layers and has two hidden layers, the first one consist of 170 node, the second one consist of 80 node, the output node uses the sigmoid activation function. The model summery is shown in Table 4.6.

4. **MLP-3 hidden layers:** this model uses the ReLU activation function in hidden layers and has three hidden layers, the first one consist of 170 node, the second one consist of 80 node, the third one consist of 40 node, the output node uses the sigmoid activation function. The model summery is shown in Table 4.7.

Table 4.4: Preceptron model summery.

| **Layer type** | **Output Shape** | **Param #** |
|---|---|---|
| Dense | (None, 1) | 343 |
| **Total params** | | 343 |
| Trainable params | | 343 |

Table 4.5: MLP-1 hidden layer model summery.

| **Layer type** | **Output Shape** | **Param #** |
|---|---|---|
| Dense | (None, 170) | 58310 |
| Dense | (None, 1) | 171 |
| **Total params** | | 58481 |
| Trainable params | | 58481 |

Table 4.6: MLP-2 hidden layers model summery.

| Layer type | Output Shape | Param # |
|---|---|---|
| Dense | (None, 170) | 58310 |
| Dense | (None, 80) | 13680 |
| Dense | (None, 1) | 81 |
| **Total params** | | 72071 |
| Trainable params | | 72071 |

Table 4.7: MLP-3 hidden layers model summery.

| Layer type | Output Shape | Param # |
|---|---|---|
| Dense | (None, 170) | 58310 |
| Dense | (None, 80) | 13680 |
| Dense | (None, 40) | 3240 |
| Dense | (None, 1) | 41 |
| **Total params** | | 75271 |
| Trainable params | | 75271 |

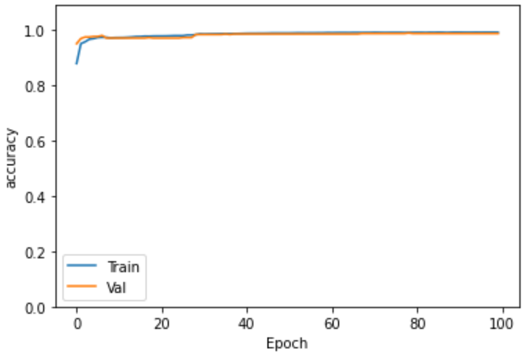
## 4.5 Results And Discussion

As recommended by data scientists we divided our dataset to 3 sets as follows:

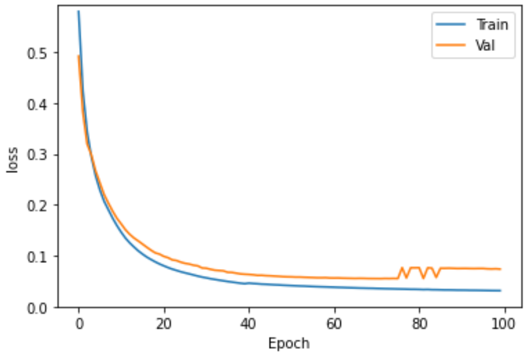
1. **Training set:** a set that consists of 4896 sample which is used to train the model.
2. **Validation set:** a set that consists of 545 sample which used to fine tune the model hyper-parameters, in our case it is used to find the optimal number of epochs.
3. **Test set:** a set that consists of 1361 which used to test the model.

To find the optimum number of epochs we train each model on 100 epochs then we draw the graph of the accuracy and loss of the model at each epoch.

Figure 4.3-a and Figure 4.3-b shows the accuracy and loss respectively over 100 epoch of the original Perceptron model, we can see from the Figure that the optimum epochs required is 60.



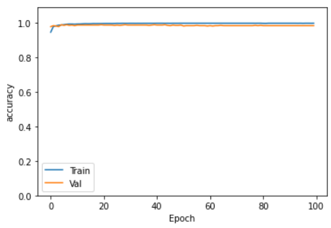

: (a) original Preceptron model accuracy by epochs



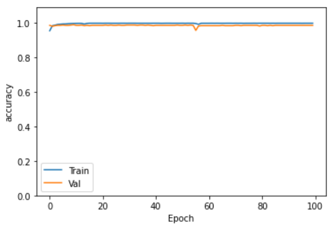

: (b) original Preceptron model loss by epochs

Figure 4.3: Original Preceptron model accuracy and loss over 100 epochs

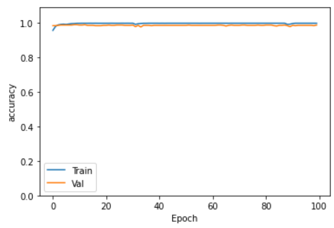
Figure 4.4-a and Figure 4.4-b and Figure 4.4-c shows the accuracy of the MLP model with 1 hidden layer, MLP model with 2 hidden layers, MLP model with 3 hidden layers respectively over 100 epoch. we can see from the Figure that it doesn't show any significant difference between the three models.



: (a) MLP with 1 hidden layer model accuracy by epochs



: (b) MLP with 2 hidden layers model accuracy by epochs

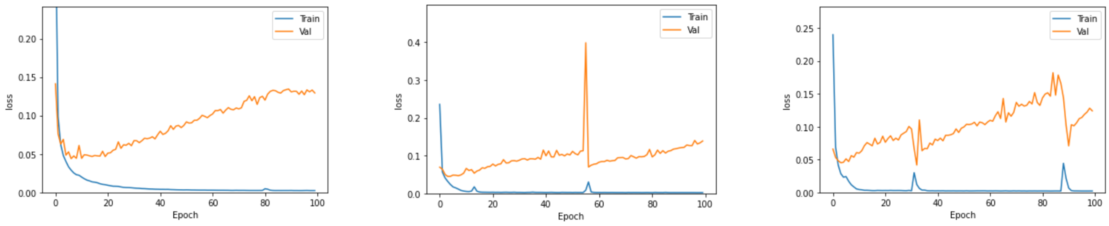


: (c) MLP with 3 hidden layers model accuracy by epochs

Figure 4.4: MLP model accuracy over 100 epochs

Figure 4.5-a and Figure 4.5-b and Figure 4.5-c shows the loss of the MLP model with 1 hidden layer, MLP model with 2 hidden layers, MLP model with 3 hidden layers respectively over 100 epoch. we can see from the figure that the optimum epochs required is 20 in each MLP model.

In our study we used two methods for each model(cross validation method & test set method), and we found that the best method for all models was cross validation. So we use 10 fold cross validation technique to evaluate our models then we calculate the average of the following metrics : Recall, Precision, Accuracy, F_Measure, FPR, TNR, FNR. the results are shown in table 4.8 and we can see




: (a) MLP with 1 hidden layer model loss by epochs

: (b) MLP with 2 hidden layers model loss by epochs

: (c) MLP with 3 hidden layers model loss by epochs

Figure 4.5: MLP model loss over 100 epochs

from the table that our Multi-layer Perceptron with 1 hidden layer model performs better then the other models in most metrics.

Table 4.8: Detailed metrics of our proposed models.

| **Model** | **Recall** | **Precision** | **Accuracy** | **F1** | **FPR** | **TNR** | **FNR** |
|---|---|---|---|---|---|---|---|
| Original Perceptron | 0.967 | **0.984** | 0.986 | 0.976 | **0.006** | **0.994** | 0.033 |
| MLP with 1 hidden layer | 0.980 | **0.984** | **0.990** | **0.982** | **0.006** | **0.994** | 0.020 |
| MLP with 2 hidden layers | **0.981** | 0.982 | **0.990** | **0.982** | 0.007 | 0.993 | **0.019** |
| MLP with 3 hidden layers | 0.978 | 0.978 | 0.988 | 0.978 | 0.009 | 0.991 | 0.022 |

The table 4.9 shows a comparison between our model with similar models that are using static analysis, we can see that our MLP with 1 hidden layer model performs better in most metrics.

Table 4.9: Our proposed models compered with similar models.

| **Reference** | **ML/DL method** | **Botnets /Benign** | **Performance** | | | |
|---|---|---|---|---|---|---|
| | | | ACC | Prec. | Rec. | F1 |
| Our method | Original Perceptron | 1929/4873 | 0.986 | **0.984** | 0.967 | 0.976 |
| Our method | MLP-1 hidden layer | 1929/4873 | **0.990** | **0.984** | 0.980 | **0.982** |
| Our method | MLP-2 hidden layers | 1929/4873 | **0.990** | 0.982 | **0.981** | **0.982** |
| Our method | MLP-3 hidden layers | 1929/4873 | 0.988 | 0.978 | 0.978 | 0.978 |
| [25] | CNN | 1929/4873 | 0.989 | 0.983 | 0.978 | 0.981 |
| [47] | CNN | 1800/3650 | 0.972 | 0.955 | 0.96 | 0.957 |
| [50] | Random Forest | 1330/1330 | 0.982 | 0.8893 | - | 0.7457 |

By comparing Perceptron model with MLP with 1 hidden layer model, we can see that Perceptron model is simple because it has only 343 trainable parameters and it achieves acceptable results, however MLP with 1 hidden layer model achieved the best results but it has 58481 trainable parameters.

## 4.6 Conclusion

In this chapter, we preformed an experiment to detect Android botnet applications based on static analysis and using Perceptron neural networks, we achieved great results compared to similar static-based methods and after comparing different models we see that Perceptron model achieved acceptable results while it is very simple, however MLP with 1 hidden layer achieved the best results.

# Conclusion

We aim in this thesis to detect Android botnet applications efficiently using a model that can achieve high results to protect Android users from botnet applications.

In this work we presented the role of smartphones in our lives, and how important their security is exponentially increasing due to the widespread of malware and botnets. Then we explained machine learning techniques and their metrics, next we provided a background on botnets and their attacks and how they affect other users/servers. after that we listed the latest methods that are being used to detect Android botnet applications, finally we performed an experimental study to detect Android botnet applications using a Perceptron models and we were able to achieve an accuracy of 99%.

For future works we can try to implement our method in real Android devices as an application that analyses every newly installed application, extract required features by reverse engineering it then classify it as benign or botnet using our trained model.

# Bibliography


[1] Mobile operating system market share worldwide from may 2020 to may 2021. URL https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed on June 13, 2021.

[2] Christiaan Beek et al. Mcafee labs threats report apr 2021. URL https://www.mcafee.com/enterprise/en-us/lp/threats-reports/apr-2021.html. Accessed on May 20, 2021.

[3] Android platform architecture. URL https://developer.android.com/guide/platform. Updated on Mar 11, 2021.

[4] Application fundamentals. URL https://developer.android.com/guide/components/fundamentals. Updated on Feb 23, 2021.

[5] The most common cyber attacks. URL https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html. Accessed on June 13, 2021.

[6] John Lehmann. Cyber crime awareness: Types of cyber attacks. URL https://www.techuseful.com/types-of-cyber-attacks/. Published on Jan 31, 2020.

[7] Jeff Melnick. Top 10 most common types of cyber attacks. URL https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/. Updated on May 18, 2021.

[8] Types of cyber attacks. URL https://www.fortinet.com/resources/cyberglossary/types-of-cyber-attacks. Accessed on June 13, 2021.

[9] Noor Qureshi. How to make sure no man-in-the-middle attack can harm you. URL https://thehacktoday.com/how-to-make-sure-no-man-in-the-middle-attack-can-harm-you/. Published on Oct 11, 2019.

[10] Alexander Amini. Introduction to deep learning. URL http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf. Published on Jun 18, 2021.

[11] Tensorflow: an end-to-end open source platform for machine learning. URL https://www.tensorflow.org/. Accessed on June 12, 2021.

[12] Alexander Amini. Introduction to deep learning. *MIT*, 6:S191, 2019.

[13] Support vector machines. URL https://scikit-learn.org/stable/modules/svm.html. Accessed on Jun 16, 2021.

[14] Explain the random forest algorithm. URL https://www.sciencedirect.com/topics/engineering/random-forest. Accessed on Jun 07, 2021.

[15] Random forest: a machine learning algorithm. URL https://qr.ae/pGMcS7. Accessed on Jun 16, 2021.

[16] R Eberhat and James Kennedy. A new optimizer using particle swarm theory. In *Sixth international symposium on micro machine and human science, Piscataway*, pages 39–43, 1995.

[17] Mahdi Moodi, Mahdieh Ghazvini, and Hossein Moodi. A hybrid intelligent approach to detect android botnet using smart self-adaptive learning-based pso-svm. *Knowledge-Based Systems*, 222:106988, 2021.

[18] Deep neural network. URL https://www.ibm.com/cloud/learn/neural-networks. Published on Aug 17, 2020.

[19] Panadda Kongsilp. Cnn: Step 3— flattening. URL https://medium.com/@PK_KwanG/cnn-step-2-flattening-50ee0af42e3e/. Published on Jul 22, 2019.

[20] Prashant Gupta. Cross-validation in machine learning. URL https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f. Published on Jun 05, 2017.

[21] Namratesh Shrivastav. Confusion matrix(tpr,fpr,fnr,tnr), precision, recall, f1-score. URL https://medium.datadriveninvestor.com/confusion-matric-tpr-fpr-fnr-tnr-precision-recall-f1-score-73efa162a25f. Published on Jan 18, 2020.

[22] Jason Brownlee. How to choose an activation function for deep learning. URL https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/. Updated on Jan 22, 2021.

[23] Cross-validation: evaluating estimator performance. URL https://scikit-learn.org/stable/modules/cross_validation.html. Accessed on July 03, 2021.

[24] Suleiman Y Yerima, Mohammed K Alzaylaee, Annette Shajan, et al. Deep learning techniques for android botnet detection. *Electronics*, 10(4):519, 2021.

[25] Suleiman Y Yerima and Mohammed K Alzaylaee. Mobile botnet detection: A deep learning approach using convolutional neural networks. In *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, pages 1–8. IEEE, 2020.

[26] Gernot Vormayr, Tanja Zseby, and Joachim Fabini. Botnet communication patterns. *IEEE Communications Surveys & Tutorials*, 19(4):2768–2796, 2017.

[27] Wei Wang, Yaoyao Shang, Yongzhong He, Yidong Li, and Jiqiang Liu. Botmark: Automated botnet detection with hybrid analysis of flow-based and graph-based

traffic behaviors. *Information Sciences*, 511:284–296, 2020.

[28] Byungha Choi, Sung-Kyo Choi, and Kyungsan Cho. Detection of mobile botnet using vpn. In *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 142–148. IEEE, 2013.

[29] David Zhao, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani, and Dan Garant. Botnet detection based on traffic behavior analysis and flow intervals. *computers & security*, 39:2–16, 2013.

[30] Fariba Haddadi, Duc Le Cong, Laura Porter, and A Nur Zincir-Heywood. On the effectiveness of different botnet detection approaches. In *International conference on information security practice and experience*, pages 121–135. Springer, 2015.

[31] What's a botnet? URL `https://www.kaspersky.com/resource-center/threats/botnet-attacks`. Accessed on June 13, 2021.

[32] Haodi Zhang, Jianye Hao, and Xiaohong Li. A method for deploying distributed denial of service attack defense strategies on edge servers using reinforcement learning. *IEEE Access*, 8:78482–78491, 2020.

[33] Eduardo Benavides, Walter Fuertes, Sandra Sanchez, and Manuel Sanchez. Classification of phishing attack solutions by employing deep learning techniques: A systematic literature review. *Developments and advances in defense and security*, pages 51–64, 2020.

[34] GS Thejas, Surya Dheeshjith, SS Iyengar, NR Sunitha, and Prajwal Badrinath. A hybrid and effective learning approach for click fraud detection. *Machine Learning with Applications*, 3:100016, 2021.

[35] Keshani Jayasinghe and Guhanathan Poravi. A survey of attack instances of cryptojacking targeting cloud infrastructure. In *Proceedings of the 2020 2nd Asia pacific information technology conference*, pages 100–107, 2020.

[36] Alicia Enterman. The matryoshka doll in russian culture. URL `https://www.macalester.edu/russian/about/resources/miscellany/matryoshka/`. Published on Dec 15, 2009.

[37] Alex Turing, Hui Wang, and Liuyang. New threat: Matryosh botnet is spreading. URL `https://blog.netlab.360.com/matryosh-botnet-is-spreading-en/`. Published on Feb 02, 2021.

[38] Pieter Arntz. Android devices caught in matryosh botnet. URL `https://blog.malwarebytes.com/malwarebytes-news/2021/02/android-devices-caught-in-matryosh-botnet/`. Published on Feb 09, 2021.

[39] Fahmida Y. Rashid. Chamois: the big botnet you didn't hear about. URL `https://duo.com/decipher/chamois-the-big-botnet-you-didnt-hear-about`. Published on Apr 09, 2019.

[40] Lihy Hay Newman. How android fought an epic botnet—and won. URL `https://www.wired.com/story/google-android-chamois-botnet`. Published on Apr 09, 2019.

[41] Sms trojan. URL https://blog.malwarebytes.com/threats/sms-trojan/. Published on June 09, 2016.

[42] Kaspersky lab quarterly report highlights botnet ddos attack growth. URL https://usa.kaspersky.com/about/press-releases/2017_kaspersky-lab-quarterly-report-highlights-botnet-ddos-attack-growth. Published on Nov 06, 2017.

[43] Jaime Cochran. The wirex botnet: How industry collaboration disrupted a ddos attack. URL https://blog.cloudflare.com/the-wirex-botnet. Published on Aug 28, 2017.

[44] Sebastian García, Maria Jose Erquiaga, and Anna Shirokova. Geost botnet. the story of the discovery of a new android banking trojan from an opsec error.

[45] Jeff Elder. Pulling back the curtain on a banking botnet. URL https://blog.avast.com/avast-researcher-helps-expose-banking-botnet-geost. Published on Oct 2, 2019.

[46] Saba Arshad, Munam A Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu. Samadroid: a novel 3-level hybrid malware detection model for android operating system. *IEEE Access*, 6:4321–4339, 2018.

[47] Sina Hojjatinia, Sajad Hamzenejadi, and Hadis Mohseni. Android botnet detection using convolutional neural networks. In *2020 28th Iranian Conference on Electrical Engineering (ICEE)*, pages 1–6. IEEE, 2020.

[48] Icsx dataset: Android botnet dataset 2015. URL https://www.unb.ca/cic/datasets/android-botnet.html. Accessed on June 10, 2021.

[49] Virustotal: A free online virus scanner. URL https://www.virustotal.com/. Accessed on June 17, 2021.

[50] Shahid Anwar, Mohamad Fadli Zolkipli, Vitaliy Mezhuyev, and Zakira Inayat. A smart framework for mobile botnet detection using static analysis. *KSII Transactions on Internet and Information Systems (TIIS)*, 14(6):2591–2611, 2020.

[51] P. H. PESKUN. Optimum monte-carlo sampling using markov chains. *Biometrika*, 60(3):607–612, 1973.

[52] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. *Proceedings of the Network and Distributed System Security Symposium 2014. San Diego, USA*, 1:15, 2014.

[53] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*, pages 95–109. IEEE, 2012.

[54] Androguard: android app reverse engineering tool. URL https://github.com/androguard/androguard. Accessed on June 08, 2021.

[55] Weka: Machine learning software in java. URL https://www.cs.waikato.ac.nz/ml/weka/. Accessed on June 08, 2021.

[56] Michelle Y Wong and David Lie. Intellidroid: A targeted input generator for the dynamic analysis of android malware. In *NDSS*, volume 16, pages 21–24, 2016.

[57] Michael Bierma, Eric Gustafson, Jeremy Erickson, David Fritz, and Yung Ryn Choe. Andlantis: Large-scale android dynamic analysis. *arXiv preprint arXiv:1410.7751*, 2014.

[58] Mahdi Moodi and Mahdieh Ghazvini. A new method for assigning appropriate labels to create a 28 standard android botnet dataset (28-sabd). *Journal of Ambient Intelligence and Humanized Computing*, 10(11):4579–4593, 2019.

[59] J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *C.,"A discrete binary version of the particle swarm algorithm," in Proc. 1997 Conf. Systems, Man, Cybernetics, Piscataway, NJ*, pages 4104–4108. IEEE, 1997.

[60] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2008.

[61] Jing J Liang and Ponnuthurai N Suganthan. Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism. In *2006 IEEE International Conference on Evolutionary Computation*, pages 9–16. IEEE, 2006.

[62] Yu Wang, Bin Li, Thomas Weise, Jianyu Wang, Bo Yuan, and Qiongjie Tian. Self-adaptive learning based particle swarm optimization. *Information Sciences*, 181(20): 4515–4538, 2011.

[63] Mahdi Moodi, Mahdieh Ghazvini, Hossein Moodi, and Behnam Ghavami. A smart adaptive particle swarm optimization–support vector machine: android botnet detection application. *The Journal of Supercomputing*, 76(12):9854–9881, 2020.

[64] Victor GT da Costa, Sylvio Barbon, Rodrigo S Miani, Joel JPC Rodrigues, and Bruno B Zarpelão. Detecting mobile botnets through machine learning and system calls analysis. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[65] Strace: Diagnostic and debugging tool for linux. URL `https://en.wikipedia.org/wiki/Strace`. Updated on Apr 26, 2021.

[66] Ahmad Karim, Victor Chang, and Ahmad Firdaus. Android botnets: A proof-of-concept using hybrid analysis approach. In *Research Anthology on Securing Mobile Technologies and Applications*, pages 75–92. IGI Global, 2021.

[67] Hyun Jae Kang, Jae-wook Jang, Aziz Mohaisen, and Huy Kang Kim. Androtracker: Creator information based android malware classification system. In *Information Security Applications-15th International Workshop, WISA*, volume 8909, page 545, 2014.

[68] Droidbox: android app dynamic analysis tool. URL `https://github.com/pjlantz/droidbox`. Accessed on June 08, 2021.

[69] Muhammad Yusof, Madihah Mohd Saudi, and Farida Ridzuan. Mobile botnet clas-

sification by using hybrid analysis. *International Journal of Engineering and Technology (UAE)*, 2018.

[70] Suleiman Yerima. Android botnet detection dataset for machine learning, Feb 2021. URL `https://figshare.com/articles/dataset/Android_botnet_detection_dataset_for_machine_learning/14079581/1`.

[71] Google colaboratory: a python development environment in your browser. URL `https://colab.research.google.com/`. Accessed on June 12, 2021.

[72] Python: a high level programming language. URL `https://www.python.org/`. Accessed on June 12, 2021.

[73] Keras: Python deep learning api. URL `https://keras.io/`. Accessed on June 12, 2021.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[75] Pandas: an open source data analysis and manipulation tool for python. URL `https://pandas.pydata.org/`. Accessed on June 12, 2021.

[76] F Rosenblat. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

[77] Andrey Kurenkov. A brief history of neural nets and deep learning. *Skynet Today*, 2020. URL `https://skynettoday.com/overviews/neural-net-history`.

# Appendices

# Used dataset features

| N° | Feature name | N° | Feature name |
|---|---|---|---|
| 1 | ACCESS_CHECKIN_PROPE-RTIES | 172 | CONNECTIVITY_CHANGE |
| 2 | ACCESS_COARSE_LOCATI-ON | 173 | UMS_CONNECTED |
| 3 | ACCESS_FINE_LOCATION | 174 | UMS_DISCONNECTED |
| 4 | ACCESS_LOCATION_EXTR-A_COMMANDS | 175 | BATTERY_LOW |
| 5 | ACCESS_MOCK_LOCATION | 176 | BATTERY_OKAY |
| 6 | ACCESS_NETWORK_STATE | 177 | BATTERY_CHANGED_ACT-ION |
| 7 | ACCESS_SURFACE_FLING-ER | 178 | INPUT_METHOD_CHANGE-D |
| 8 | ACCESS_WIFI_STATE | 179 | SIG_STR |
| 9 | ACCOUNT_MANAGER | 180 | SIM_FULL |
| 10 | ADD_VOICEMAIL | 181 | SEND_MESSAGE |
| 11 | AUTHENTICATE_ACCOUN-TS | 182 | UID_REMOVED |
| 12 | BATTERY_STATS | 183 | CAMERA_BUTTON |
| 13 | BIND_ACCESSIBILITY_SER-VICE | 184 | .zip |
| 14 | BIND_APPWIDGET | 185 | .apk |
| 15 | BIND_DEVICE_ADMIN | 186 | .dex |
| 16 | BIND_INPUT_METHOD | 187 | .exe |
| 17 | BIND_REMOTEVIEWS | 188 | .so |
| 18 | BIND_TEXT_SERVICE | 189 | abortBroadcast |
| 19 | BIND_VPN_SERVICE | 190 | HttpPost.*init |
| 20 | BIND_WALLPAPER | 191 | HttpGet.*init |
| 21 | BLUETOOTH | 192 | HttpUriRequest |
| 22 | BLUETOOTH_ADMIN | 193 | setRequestMethod( |
| 23 | BRICK | 194 | getInputStream( |
| 24 | BROADCAST_PACKAGE_R-EMOVED | 195 | getOutputStream( |
| 25 | BROADCAST_SMS | 196 | Ljava.net.URLDecoder |
| 26 | BROADCAST_STICKY | 197 | System.*loadLibrary |
| 27 | BROADCAST_WAP_PUSH | 198 | Ljava\/lang\/Object.*getClass |
| 28 | CALL_PHONE | 199 | Ljava\/lang\/Class.*getMethod-s( |

| N° | Feature name | N° | Feature name |
|---|---|---|---|
| 29 | CALL_PRIVILEGED | 200 | Ljava\/lang\/Class.*forName |
| 30 | CAMERA | 201 | Ljava\/lang\/Class.*cast |
| 31 | CHANGE_COMPONENT_EN-ABLED_STATE | 202 | Ljava\/lang\/Class.*getClasses |
| 32 | CHANGE_CONFIGURATION | 203 | Ljava\/lang\/Class.*getCanoni-calName |
| 33 | CHANGE_NETWORK_STA-TE | 204 | Ljava\/lang\/Class.*getDeclare-dClasses |
| 34 | CHANGE_WIFI_MULTICAS-T_STATE | 205 | Ljava\/lang\/Class.*getDeclare-dField |
| 35 | CHANGE_WIFI_STATE | 206 | Ljava\/lang\/Class.*getField |
| 36 | CLEAR_APP_CACHE | 207 | Ljava\/lang\/Class.*getSigners |
| 37 | CLEAR_APP_USER_DATA | 208 | Ljava\/lang\/Class.*getResourc-e |
| 38 | CONTROL_LOCATION_UP-DATES | 209 | Ljava\/lang\/Class.*getPackage |
| 39 | DELETE_CACHE_FILES | 210 | DexClassLoader |
| 40 | DELETE_PACKAGES | 211 | DexFile.*loadClass |
| 41 | DEVICE_POWER | 212 | DexFile.*getName |
| 42 | DIAGNOSTIC | 213 | DexFile.*loadDex |
| 43 | DISABLE_KEYGUARD | 214 | ClassLoader |
| 44 | DUMP | 215 | findClass |
| 45 | EXPAND_STATUS_BAR | 216 | defineClass |
| 46 | FACTORY_TEST | 217 | PathClassLoader |
| 47 | FLASHLIGHT | 218 | URLClassLoader |
| 48 | FORCE_BACK | 219 | loadClass( |
| 49 | GET_ACCOUNTS | 220 | android.os.IBinder |
| 50 | GET_PACKAGE_SIZE | 221 | getCallingUid( |
| 51 | GET_TASKS | 222 | getCallingPid( |
| 52 | GLOBAL_SEARCH | 223 | transact( |
| 53 | HARDWARE_TEST | 224 | onBind |
| 54 | INJECT_EVENTS | 225 | IRemoteService |
| 55 | INSTALL_LOCATION_PRO-VIDER | 226 | ServiceConnection |
| 56 | INSTALL_PACKAGES | 227 | Context.bindService |
| 57 | INTERNAL_SYSTEM_WIND-OW | 228 | bindService |
| 58 | INTERNET | 229 | IBinder |
| 59 | KILL_BACKGROUND_PRO-CESSES | 230 | Binder |

| N° | Feature name | N° | Feature name |
|---|---|---|---|
| 60 | MANAGE_ACCOUNTS | 231 | getBinder |
| 61 | MANAGE_APP_TOKENS | 232 | MessengerService |
| 62 | MASTER_CLEAR | 233 | onServiceConnected( |
| 63 | MODIFY_AUDIO_SETTINGS | 234 | Ljavax\/crypto\/Cipher |
| 64 | MODIFY_PHONE_STATE | 235 | Ljavax\/crypto\/spec\/SecretK-eySpec |
| 65 | MOUNT_FORMAT_FILESY-STEMS | 236 | SecretKey |
| 66 | MOUNT_UNMOUNT_FILES-YSTEMS | 237 | KeySpec |
| 67 | NFC | 238 | doFinal( |
| 68 | PERSISTENT_ACTIVITY | 239 | Runtime.*exec |
| 69 | PROCESS_OUTGOING_CAL-LS | 240 | createSubprocess |
| 70 | READ_CALENDAR | 241 | Runtime.*load |
| 71 | READ_CALL_LOG | 242 | Runtime.*loadLibrary |
| 72 | READ_CONTACTS | 243 | ProcessBuilder |
| 73 | READ_EXTERNAL_STORA-GE | 244 | Process.*start |
| 74 | READ_FRAME_BUFFER | 245 | Process.*myPid |
| 75 | READ_HISTORY_BOOKMA-RKS | 246 | Runtime.*getRuntime |
| 76 | READ_INPUT_STATE | 247 | killProcess( |
| 77 | READ_LOGS | 248 | android.telephony.gsm.SmsMan-ager |
| 78 | READ_PHONE_STATE | 249 | android.telephony.SmsManager |
| 79 | READ_PROFILE | 250 | divideMessage |
| 80 | READ_SMS | 251 | sendTextMessage( |
| 81 | READ_SOCIAL_STREAM | 252 | android.content.pm.PackageInfo |
| 82 | READ_SYNC_SETTINGS | 253 | android.content.pm.Signature |
| 83 | READ_SYNC_STATS | 254 | PackageInstaller |
| 84 | READ_USER_DICTIONARY | 255 | getInstalledPackages( |
| 85 | REBOOT | 256 | TelephonyManager.*getDeviceId |
| 86 | RECEIVE_BOOT_COMPLE-TED | 257 | TelephonyManager.*getSubscri-berId |
| 87 | RECEIVE_MMS | 258 | TelephonyManager.*getSimSeri-alNumber |
| 88 | RECEIVE_SMS | 259 | TelephonyManager.*getLine1N-umber |

| N° | Feature name | N° | Feature name |
|---|---|---|---|
| 89 | RECEIVE_WAP_PUSH | 260 | TelephonyManager.*getNetwor-kOperator |
| 90 | RECORD_AUDIO | 261 | TelephonyManager.*getSimOpe-rator |
| 91 | REORDER_TASKS | 262 | TelephonyManager.*getCallStat-e |
| 92 | RESTART_PACKAGES | 263 | TelephonyManager.*isNetwork-Roaming |
| 93 | SEND_SMS | 264 | getCellLocation( |
| 94 | SET_ACTIVITY_WATCHER | 265 | TelephonyManager.*getSimCou-ntryIso |
| 95 | SET_ALARM | 266 | Ljava.util.Timer |
| 96 | SET_ALWAYS_FINISH | 267 | Ljava.util.Timer.*schedule |
| 97 | SET_ANIMATION_SCALE | 268 | Ljava.util.TimerTask |
| 98 | SET_DEBUG_APP | 269 | Ljava.util.Date |
| 99 | SET_ORIENTATION | 270 | AssetManager |
| 100 | SET_POINTER_SPEED | 271 | getResources |
| 101 | SET_PREFERRED_APPLIC-ATIONS | 272 | Landroid.content.res.AssetMana-ger |
| 102 | SET_PROCESS_LIMIT | 273 | getAssets |
| 103 | SET_TIME | 274 | getContentResolver.*query |
| 104 | SET_TIME_ZONE | 275 | content:\/\/sms |
| 105 | SET_WALLPAPER | 276 | content:\/\/telephony |
| 106 | SET_WALLPAPER_HINTS | 277 | content:\/\/mail |
| 107 | SIGNAL_PERSISTENT_PRO-CESSES | 278 | content:\/\/downloads |
| 108 | STATUS_BAR | 279 | content:\/\/browser |
| 109 | SUBSCRIBED_FEEDS_READ | 280 | content:\/\/contacts |
| 110 | SUBSCRIBED_FEEDS_WRI-TE | 281 | Ljava.net.InetSocketAddress |
| 111 | SYSTEM_ALERT_WINDOW | 282 | getDataDir( |
| 112 | UPDATE_DEVICE_STATS | 283 | getApplicationInfo( |
| 113 | USE_CREDENTIALS | 284 | getSystemService( |
| 114 | USE_SIP | 285 | BatteryManager |
| 115 | VIBRATE | 286 | AudioManager |
| 116 | WAKE_LOCK | 287 | CameraManager |
| 117 | WRITE_APN_SETTINGS | 288 | NfcManager |
| 118 | WRITE_CALENDAR | 289 | SensorManager |
| 119 | WRITE_CALL_LOG | 290 | UsbManager |
| 120 | WRITE_CONTACTS | 291 | WifiManager |

| N° | Feature name | N° | Feature name |
|---|---|---|---|
| 121 | WRITE_EXTERNAL_STOR-AGE | 292 | BluetoothManager |
| 122 | WRITE_GSERVICES | 293 | addFlags( |
| 123 | WRITE_HISTORY_BOOKM-ARKS | 294 | setFlags( |
| 124 | WRITE_PROFILE | 295 | getRunningServices( |
| 125 | WRITE_SECURE_SETTINGS | 296 | getMemoryInfo( |
| 126 | WRITE_SETTINGS | 297 | restartPackage( |
| 127 | WRITE_SMS | 298 | onActivityResult |
| 128 | WRITE_SOCIAL_STREAM | 299 | getNetworkInfo( |
| 129 | WRITE_SYNC_SETTINGS | 300 | getExtraInfo( |
| 130 | WRITE_USER_DICTIONARY | 301 | getTypeName( |
| 131 | android.intent.action.TIME_SE-T | 302 | isConnected( |
| 132 | android.intent.action.TIMEZO-NE_CHANGED | 303 | getState( |
| 133 | android.intent.action.BOOT_C-OMPLETED | 304 | setWifiEnabled( |
| 134 | android.intent.action.PACKAG-E_ADDED | 305 | getWifiState( |
| 135 | android.intent.action.PACKAG-E_CHANGED | 306 | android.os.Handler |
| 136 | android.intent.action.PACKAG-E_REMOVED | 307 | obtainMessage( |
| 137 | android.intent.action.PACKAG-E_RESTARTED | 308 | sendMessage( |
| 138 | android.intent.action.PACKAG-E_DATA_CLEARED | 309 | DataInputStream.*available( |
| 139 | android.intent.action.UID_RE-MOVED | 310 | FileOutputStream.*write( |
| 140 | android.intent.action.ACTION-_POWER_CONNECTED | 311 | io.File.*delete( |
| 141 | android.intent.action.ACTION-_POWER_DISCONNECTED | 312 | io.File.*mkdir |
| 142 | android.intent.action.ACTION-_SHUTDOWN | 313 | io.File.*exists( |
| 143 | android.intent.action.PACKAG-E_REPLACED | 314 | ZipInputStream.*read( |
| 144 | android.intent.action.BATTER-Y_LOW | 315 | ZipInputStream.*close( |

| N° | Feature name | N° | Feature name |
|---|---|---|---|
| 145 | android.intent.action.BATTER-Y_OKAY | 316 | ZipInputStream.*getNextEntry( |
| 146 | android.intent.action.CALL | 317 | getElementByTagName( |
| 147 | android.intent.action.CALL_B-UTTON | 318 | getAttribute( |
| 148 | android.intent.action.CAMERA-_BUTTON | 319 | getDocumentElement( |
| 149 | android.intent.action.NEW_O-UTGOING_CALL | 320 | Landroid/location/LocationMan-ager.*getAllProviders( |
| 150 | android.intent.action.REBOOT | 321 | android.hardware |
| 151 | android.intent.action.SCREEN-_OFF | 322 | checkSignatures( |
| 152 | android.intent.action.SCREEN-_ON | 323 | getSystemAvailableFeatures( |
| 153 | android.intent.action.SEND | 324 | chmod |
| 154 | android.intent.action.SENDTO | 325 | chown |
| 155 | android.intent.action.SET_WA-LLPAPER | 326 | \/system\/app |
| 156 | android.settings.NETWORK_-OPERATOR_SETTINGS | 327 | \/system\/bin |
| 157 | intent.action.RUN | 328 | \/system\/bin\/su |
| 158 | android.intent.action.SEND_M-ULTIPLE | 329 | \/system\/bin\/sh |
| 159 | android.settings.APN_SETTIN-GS | 330 | mount |
| 160 | NEW_OUTGOING_CALL | 331 | remount |
| 161 | USER_PRESENT | 332 | grep |
| 162 | SMS_RECEIVED | 333 | \/sh |
| 163 | PACKAGE_REPLACED | 334 | \/bin |
| 164 | PACKAGE_INSTALL | 335 | insmod |
| 165 | ACTION_MAIN | 336 | stdout |
| 166 | SEND_MULTIPLE | 337 | stderr |
| 167 | settings.APN_SETTINGS | 338 | killall |
| 168 | wifi.WIFI_STATE_CHANGED | 339 | reboot |
| 169 | PICK_WIFI_WORK | 340 | \/dev\/net |
| 170 | PHONE_STATE | 341 | \/system |
| 171 | WAP_PUSH_RECEIVED | 342 | pminstall |